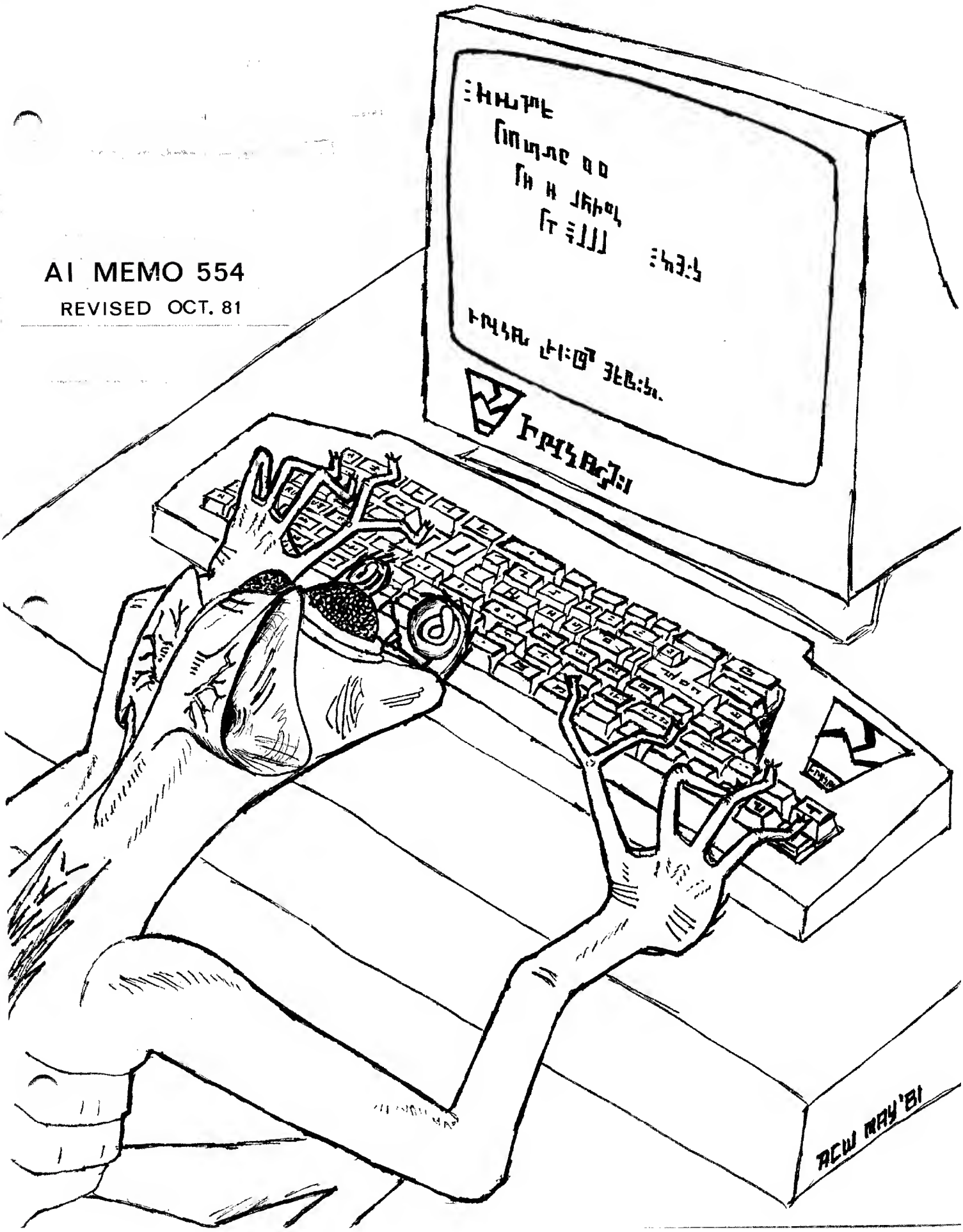
AI MEMO 554

REVISED OCT. 81



MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

AI Memo 554

22 October 1981

EMACS Manual for ITS Users

by

Richard M. Stallman

A reference manual

for the extensible, customizable, self-documenting

real-time display editor

This manual corresponds to EMACS version 162

This report describes work done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505.

9. Killing and Moving Text 35

9.1. Deletion and Killing 35
9.2. Un-Killing 37
9.3. Other Ways of Copying Text 39

10. Searching 41

11. Commands for English Text 45

11.1. Word Commands 45
11.2. Sentence and Paragraph Commands 46
11.3. Indentation Commands for Text 48
11.4. Text Filling 50
11.5. Case Conversion Commands 51
11.6. Font-Changing 52
11.7. Underlining 53
11.8. SCRIBE Mode 53
11.9. Dissociated Press 55

12. Commands for Fixing Typos 57

12.1. Killing Your Mistakes 57
12.2. Transposition 58
12.3. Case Conversion 58
12.4. Checking and Correcting Spelling 59

13. File Handling 61

13.1. Visiting Files 61
13.2. How to Undo Drastic Changes to a File 63
13.3. Auto Save Mode: Protection Against Disasters 63
13.4. Listing a File Directory 65
13.5. Cleaning a File Directory 66
13.6. DIRED, the Directory Editor Subsystem 66
13.7. Miscellaneous File Operations 68
13.8. The Directory Comparison Subsystem 69

14. Using Multiple Buffers 71

14.1. Creating and Selecting Buffers 71
14.2. Using Existing Buffers 72
14.3. Killing Buffers 73

15. Controlling the Display 75

16. Two Window Mode 77

16.1. Multiple Windows and Multiple Buffers 78

17. Narrowing 81

24.5. Reporting Bugs 140

25. Word Abbreviation Input 143

25.1. Basic Usage 144
25.2. Advanced Usage 147
25.3. Teco Details for Extension Writers 150

26. The PICTURE Subsystem, an Editor for Text Pictures 151

27. Sorting Functions 153

Appendix I. Particular Types of Terminals 155

I.1. Ideal Keyboards 155
I.2. Keyboards with an "Edit" key 156
I.3. ASCII Keyboards 156
I.4. Upper-case-only Terminals 157
I.5. The SLOWLY Package for Slow Terminals 158

Appendix II. Use of EMACS from Printing Terminals 161

Distribution of EMACS 163

Glossary 165

Command Summary 175

Catalog of Libraries 195

Index of Variables 199

EMACS Command Chart (as of 4/06/81) 209

Index 217

Preface

This manual documents the use and simple customization of the display editor EMACS with the ITS operating system. The reader is *not* expected to be a programmer. Even simple customizations do not require programming skill, but the user who is not interested in customizing can ignore the scattered customization hints.

This is primarily a reference manual, but can also be used as a primer. However, I recommend that the newcomer first use the on-line, learn-by-doing tutorial TEACHEMACS, by typing :TEACHEMACS<cr> while in HACTRN. With it, you learn EMACS by using EMACS on a specially designed file which describes commands, tells you when to try them, and then explains the results you see. This gives a more vivid introduction than a printed manual.

On first reading, you need not make any attempt to memorize chapters 1 and 2, which describe the notational conventions of the manual and the general appearance of the EMACS display screen. It is enough to be aware of what questions are answered in these chapters, so you can refer back when you later become interested in the answers. After reading the Basic Editing chapter you should practice the commands there. The next few chapters describe fundamental techniques and concepts that are referred to again and again. It is best to understand them thoroughly, experimenting with them if necessary.

To find the documentation on a particular command, look in the index if you know what the command is. Both command characters and function names are indexed. If you know vaguely what the command does, look in the command summary. The command summary contains a line or two about each command, and a cross reference to the section of the manual that describes the command in more detail; related commands are grouped together. There is also a glossary, with a cross reference for each term.

Many user-contributed libraries accompany EMACS, and often then have no documentation except their on-line self-documentation. Browsing through the catalogue of libraries in this manual is a good way to find out what is available.

This manual has in two versions, one for ITS, and one for Twenex, DEC's "TOPS-20" system. Each version comes in three forms: the published form, the LPT form, and the INFO form. You can order the published form from the Publications Department of the Artilicial Intelligence lab for $3.25 per copy; be sure to specify the ITS version or the Twenex version. The LPT form is available on line as EMACS; EMACS GUIDE for printing on unsophisticated hard copy devices such as terminals and line printers. The INFO form is for on-line perusal with the INFO program. All three forms are substantially the same.

Introduction

You are about to read about EMACS, an advanced, self-documenting, customizable, extensible real-time display editor.

We say that EMACS is a display editor because normally the text being edited is visible on the screen and is updated automatically as you type your commands. See section 1 [Display], page 5.

We call it a real-time editor because the display is updated very frequently, usually after each character or pair of characters you type. This minimizes the amount of information you must keep in your head as you edit. See section 3 [Basic], page 13.

We call EMACS advanced because it provides facilities that go beyond simple insertion and deletion: filling of text; automatic indentation of programs; viewing two files at once; and dealing in terms of characters, words, lines, sentences, paragraphs, and pages, as well as expressions and comments in several different programming languages. It is much easier to type one command meaning "go to the end of the paragraph" than to find the desired spot with repetition of simpler commands.

Self-documenting means that at any time you can type a special character, the "Help" key, to find out what your options are. You can also use it to find out what any command does, or to find all the commands that pertain to a topic. See section 7 [Help], page 29.

Customizable means that you can change the definitions of EMACS commands in little ways. For example, if you use a programming language in which comments start with <** and end with **>, you can tell the EMACS comment manipulation commands to use those strings. Another sort of customization is rearrangement of the command set. For example, if you prefer the four basic cursor motion commands (up, down, left and right) on keys in a diamond pattern on the keyboard, you can have it. See section 21.8 [Customization], page 114.

Extensible means that you can go beyond simple customization and write entirely new commands, programs in the language TECO. EMACS is an "on-line extensible" system, which means that it is divided into many functions that call each other, any of which can be redefined in the middle of an editing session. Any part of EMACS can be replaced without making a separate copy of all of EMACS. Many already written extensions are distributed with EMACS, and some (including DIRED, PAGE, PICTURE, SORT, TAGS, and WORDAB) are documented in this manual. Although only a programmer can write an extension, anybody can use it afterward.

Extension requires programming in TECO, a rather obscure language. If you are clever and bold, you might wish to learn how. See the file INFO;CONV >, for advice on learning TECO. This manual does not even try to explain how to write TECO programs, but it does contain some notes that are useful primarily to the extension writer

Chapter One

The Organization of the Screen

EMACS divides the screen into several areas, each of which contains its own sorts of information. The biggest area, of course, is the one in which you usually see the text you are editing. The terminal's cursor usually appears in the middle of the text, showing the position of *point*, the location at which editing takes place. While the cursor appears to point *at* a character, point should be thought of as *between* two characters; it points *before* the character that the cursor appears on top of. Terminals have only one cursor, and when output is in progress it must appear where the typing is being done. This does not mean that point is moving. It is only that EMACS has no way to show you the location of point except when the terminal is idle.

The top lines of the screen are usually available for text but are sometimes pre-empted by an *error message*, which says that some command you gave was illegal or used improperly, or by *typeout* from a command (such as, a listing of a file directory). Error messages are typically one line, end with a question mark, and are accompanied by ringing the bell. Typeout generally has none of those characteristics.

The error message or typeout appears there for your information, but it is not part of the file you are editing, and it goes away if you type any command. If you want to make it go away immediately but not do anything else, you can type a Space. (Usually a Space inserts itself, but when there is an error message or typeout on the screen it does nothing but get rid of that.) The terminal's cursor always appears at the end of the error message or typeout, but this does not mean that point has moved. The cursor moves back to the location of point after the error message or typeout goes away.

If you type a question mark when an error message is on the screen, you enter the EMACS error handler. You probably don't want to do this unless you know how to write TECO programs. If you do it by accident, C-] (the standard abort character) will get you out. Enough said.

A few lines at the bottom of the screen compose what is called the *echo area*. The variable Echo Area Height controls how many lines long it is. *Echoing* means printing out the commands that you type. EMACS commands are usually not echoed at all, but if you pause for more than a second in the middle of a multi-character command then all the characters typed so far are echoed. This is intended to *prompt* you for the rest of the command. The rest of the command is echoed, too, as you type it. This behavior is designed to give confident users optimum response, while giving hesitant users maximum feedback.

EMACS also uses the echo area for reading and displaying the arguments for some

is on. "Ovwrt" means that Overwrite mode is on. See section 22.1 [Minor Modes], page 115, for more information. "Def" means that a keyboard macro is being defined; although this is not exactly a minor mode, it is still useful to be reminded about. See section 22.8 [Keyboard Macros], page 128. "Narrow" means that editing is currently restricted to only a part of the buffer. See section 17 [Narrowing], page 81.

bfr is the name of the currently selected *buffer*. Each buffer has its own name and holds a file being edited; this is how EMACS can hold several files at once. But at any time you are editing only one of them, the *selected* buffer. When we speak of what some command does to "the buffer", we are talking about the currently selected buffer. Multiple buffers make it easy to switch around between several files, and then it is very useful that the mode line tells you which one you are editing at any time. However, before you learn how to use multiple buffers, you will always be in the buffer called "Main", which is the only one that exists when EMACS starts up. If the name of the buffer is the same as the first name of the file you are visiting, then the buffer name is left out of the mode line. See section 14 [Buffers], page 71, for how to use more than one buffer in one EMACS.

file is the name of the file that you are editing. It is the last file that was visited in the buffer you are in. It is followed, in parentheses, by the file *version number* most recently visited or saved provided the second filename visited is ">", so that version numbers are allowed. "(R-O)" after the filename means that the file or buffer is *read-only*; a file visited read-only will not be saved unless you insist, while a read-only buffer does not allow you to alter its contents at all. See section 13.1 [Visiting], page 61, for more information.

The star at the end of the mode line means that there are changes in the buffer that have not been saved in the file. If the file has not been changed since it was read in or saved, there is no star.

pos tells you whether there is additional text above the top of the screen, or below the bottom. If your file is small and it is all on the screen, --pos-- is omitted. Otherwise, it is --TOP-- if you are looking at the beginning of the file, --BOT-- if you are looking at the end of the file, or --nn%-- where nn is the percentage of the file above the top of the screen.

Sometimes you will see --MORE-- instead of --nn%--. This happens when typeout from a command is too long to fit on the screen. It means that if you type a Space the next screenful of information will be printed. If you are not interested, typing anything but a Space will cause the rest of the output to be discarded. Typing a Rubout will discard the output and do nothing else. Typing any other command will discard the rest of the output and also do the command. When the output is discarded, "FLUSHED" is printed after the --MORE--.

If you are accustomed to other display editors, you may be surprised that EMACS does not always display the page number and line number of point in the mode line. This is because the text is stored in a way that makes it difficult to compute this information. Displaying them all the time would be too slow to be borne. When you want to know the page and line number of point, you must ask for the information with the M-X What Page command. See section 18 [Pages], page 83. However, once you are adjusted to EMACS, you will rarely have any reason to be concerned with page numbers or line numbers.

Chapter Two

Character Sets and Command Input Conventions

In this chapter we introduce the terminology and concepts used to talk about EMACS commands. EMACS is designed to be used with a kind of keyboard with two special shift keys which can type 512 different characters, instead of the 128 different characters which ordinary ASCII keyboards can send. The terminology of EMACS commands is formulated in terms of these shift keys. So that EMACS can be used on ASCII terminals, we provide two-character ASCII circumlocutions for the command characters which are not ASCII.

2.1. The 9-bit Command Character Set

EMACS is designed ideally to be used with terminals whose keyboards have a pair of shift keys, labeled "Control" and "Meta", either or both of which can be combined with any character that you can type. These shift keys produce *Control* characters and *Meta* characters, which are the editing commands of EMACS. We name each of these characters by prefixing "Control-" (or "C-"), "Meta-" (or "M-") or both to the basic character: thus, Meta-F or M-F is the character which is F typed with the Meta key held down. C-M-; is the Semicolon character with both the Control and Meta keys. Control in the EMACS command character set is not precisely the same as Control in the ASCII character set, but the general purpose is the same.

There are 128 basic characters. Multiplied by the four possibilities of the Control and Meta keys, this makes 512 characters in the EMACS command character set. So it is called the 512-character set, to distinguish it from ASCII, which has only 128 characters. It is also called the *9-bit* character set because 9 bits are required to express a number from 0 to 511. Note that the 512-character set is used only for keyboard commands. Characters in files being edited with EMACS are ASCII characters.

Sadly, most terminals do not have ideal EMACS keyboards. In fact, the only ideal keyboards are at MIT. On nonideal keyboards, the Control key is somewhat limited (it can only be combined with some characters, not with all), and the Meta key may not exist at all. We make it possible to use EMACS on a nonideal terminal by providing two-character circumlocutions, made up of ASCII characters that you can type, for the characters that you can't type. These circumlocutions start with a *bit prefix* character; see below. For example, to use the Meta-A command, you could type

typed as two characters on any terminal. You can create new prefix characters when you customize. See the file INFO;CONV >, node Prefix.

2.3. Commands, Functions, and Variables

Most of the EMACS commands documented herein are members of this 9-bit character set. Others are pairs of characters from that set. However, EMACS doesn't really implement commands directly. Instead, EMACS is composed of *functions*, which have long names such as "^R Down Real Line" and definitions which are programs that perform the editing operations. Then *commands* such as C-N are connected to functions through the *command dispatch table*. When we say that C-N moves the cursor down a line, we are glossing over a distinction which is unimportant for ordinary use, but essential for customization: it is the function ^R Down Real Line which knows how to move down a line, and C-N moves down a line *because* it is connected to that function. We usually ignore this subtlety to keep things simple. To give the extension-writer the information he needs, we state the name of the function which really does the work in parentheses after mentioning the command name. For example: "C-N (^R Down Real Line) moves the cursor down a line". In the EMACS wall chart, the function names are used as a form of very brief documentation for the command characters. See section 5.2 [Functions], page 21.

The "^R " which appears at the front of the function name is simply part of the name. By convention, a certain class of functions have names which start with "^R ".

While we are on the subject of customization information which you should not be frightened of, it's a good time to tell you about *variables*. Often the description of a command will say "to change this, set the variable Mumble Foo". A variable is a name used to remember a value. EMACS contains many variables which are there so that you can change them if you want to customize. The variable's value is examined by some command, and changing the value makes the command behave differently. Until you are interested in customizing, you can ignore this information. When you are ready to be interested, read the basic information on variables, and then the information on individual variables will make sense. See section 22.3 [Variables], page 118.

2.4. Notational Conventions for ASCII Characters

Control characters in files, your EMACS buffer, or TECO programs, are ordinary ASCII characters. The special 9-bit character set applies only to typing EMACS commands. ASCII contains the printing characters, rubout, and some control characters. Most ASCII control characters are represented in this manual as uparrow or caret followed by the corresponding non-control character: control-E is represented as ↑E.

Some ASCII characters have special names. These include tab (011), backspace (010), linefeed (012), return (015), altmode (033), space (040), and rubout (177). To make it clear whether we are talking about a 9-bit character or an ASCII character, we

Chapter Three

Basic Editing Commands

We now give the basics of how to enter text, make corrections, and save the text in a file. If this material is new to you, you might learn it more easily by running the :TEACHEMACS program.

3.1. Inserting Text

To insert printing characters into the text you are editing, just type them. When EMACS is at top level, all printing characters you type are inserted into the text at the cursor (that is, at *point*), and the cursor moves forward. Any characters after the cursor move forward too. If the text in the buffer is FOOBAR, with the cursor before the B, then if you type XX, you get FOOXXBAR, with the cursor still before the B.

To correct text you have just inserted, you can use Rubout. Rubout deletes the character *before* the cursor (not the one that the cursor is on top of or under; that is the character *after* the cursor). The cursor and all characters after it move backwards. Therefore, if you type a printing character and then type Rubout, they cancel out.

To end a line and start typing a new one, type Return (Customizers, note: this runs the function ^R CRLF). Return operates by inserting a line separator, so if you type Return in the middle of a line, you break the line in two. Return really inserts two characters, a carriage return and a linefeed (a CRLF), but almost everything in EMACS makes them look like just one character, which you can think of as a line-separator character. For example, typing Rubout when the cursor is at the beginning of a line rubs out the line separator before the line, joining that line with the preceding line.

If you add too many characters to one line, without breaking it with a Return, the line will grow to occupy two (or more) lines on the screen, with a "!" at the extreme right margin of all but the last of them. The "!" says that the following screen line is not really a distinct line in the file, but just the *continuation* of a line too long to fit the screen.

Direct insertion works for printing characters and space, but other characters act as editing commands and do not insert themselves. If you need to insert a control character, Altmode, Tab or Rubout, you must *quote* it by typing the Control-Q (^R Quoted Insert) command first. See section 2 [Characters], page 9. Inserting a ↑Z is harder because EMACS cannot even receive the character; you must use the minibuffer as in Altmode Altmode 26i Altmode Altmode. See section 23 [Minibuffer], page 131.

file FOO is not really changed. If the file FOO doesn't exist, and you want to create it, visit it as if it did exist. When you save your text with C-X C-S the file will be created.

Of course, there is a lot more to learn about using files. See section 13 [Files], page 61.

3.5. Help

If you forget what a command does, you can find out with the Help character. The Help character is Top-H if you have a Top key, or Control-_ H (two characters!) otherwise. Type Help followed by C and the command you want to know about. Help can help you in other ways as well. See section 7 [Help], page 29.

3.6. Using Blank Lines Can Make Editing Faster

C-O	Insert one or more blank lines after the cursor.
C-X C-O	Delete all but one of many consecutive blank lines.

It is much more efficient to insert text at the end of a line than in the middle. So if you want to stick a new line before an existing one, the best way is to make a blank line there first and then type the text into it, rather than inserting the new text at the beginning of the existing line and finally inserting a line separator. Making the blank line first also makes the meaning of the text clearer while you are typing it in.

To make a blank line, you can type Return and then C-B. But there is a single character for this: C-O (Customizers: this is the built-in function ^R Open Line). So, FOO Return is equivalent to C-O FOO.

If you want to insert many lines, you can type many C-O's at the beginning (or you can give C-O an argument to tell it how many blank lines to make. See section 4 [Arguments], page 17, for how). As you then insert lines of text, you will notice that Return behaves strangely: it "uses up" the blank lines instead of pushing them down.

If you don't use up all the blank lines, you can type C-X C-O (the function ^R Delete Blank Lines) to get rid of all but one. When point is on a blank line, C-X C-O replaces all the blank lines around that one with a single blank line. When point is on a nonblank line, C-X C-O deletes any blank lines following that nonblank line.

Chapter Four

Giving Numeric Arguments to EMACS Commands

Any EMACS command can be given a *numeric argument*. Some commands interpret the argument as a repetition count. For example, giving an argument of ten to the C-F command (move forward one character) moves forward ten characters. With these commands, no argument is equivalent to an argument of 1.

Some commands care only about whether there is an argument, and not about its value; for example, the command M-Q (^R Fill Paragraph) with no arguments fills text, but with an argument justifies the text as well.

Some commands use the value of the argument, but do something peculiar when there is no argument. For example, the C-K (^R Kill Line) command with an argument <n> kills <n> lines and the line separators that follow them. But C-K with no argument is special; it kills the text up to the next line separator, or, if point is right at the end of the line, it kills the line separator itself. Thus, two C-K commands with no arguments can kill a nonblank line, just like C-K with an argument of one.

The fundamental way of specifying an argument is to use the C-U (^R Universal Argument) command followed by the digits of the argument. Negative arguments are allowed. Often they tell a command to move or act backwards. A negative argument is entered with C-U followed by a minus sign and the digits of the value of the argument.

C-U followed by a character which is neither a digit nor a minus sign has the special meaning of "multiply by four". It multiplies the argument for the next command by four. Two such C-U's multiply it by sixteen. Thus, C-U C-U C-F moves forward sixteen characters. This is a good way to move forward "fast", since it moves about 1/4 of a line on most terminals. Other useful combinations are C-U C-N, C-U C-U C-N (move down a good fraction of a screen), C-U C-U C-O (make "a lot" of blank lines), and C-U C-K (kill four lines). With commands like M-Q that care whether there is an argument but not what the value is, C-U is a good way of saying "I want an argument".

A few commands treat a plain C-U differently from an ordinary argument. A few others may treat an argument of just a minus sign differently from an argument of -1. These unusual cases will be described when they come up; they are always for reasons of convenience of use.

There are other, terminal-dependent ways of specifying arguments. They have the same effect but may be easier to type. See the appendix. If your terminal has a

Chapter Five

Extended (Meta-X) Commands and Functions

Not all EMACS commands are of the one or two character variety you have seen so far. Most commands have long names composed of English words. This is for two reasons: the long names are easier to remember and more suggestive, and there are not enough two-character combinations for every command to have one.

The commands with long names are known as *extended commands* because they extend the set of two-character commands.

5.1. Issuing Extended Commands

M-X	Begin an extended command. Follow by command name and arguments.
C-M-X	Begin an extended command. Follow by the command name only; the command will ask for any arguments.
C-X Altmode	Re-execute recent extended command.

Extended commands are also called *M-X commands*, because they all start with the character Meta-X (^R Extended Command). The M-X is followed by the command's long, suggestive name, actually the name of a function to be called. Terminate the name of the function with a Return (unless you are supplying string arguments; see below). For example, Meta-X Auto Fill Mode<cr> invokes the function Auto Fill Mode. This function when executed turns Auto Fill mode on or off.

We say that M-X Foo<cr> "calls the function Foo". When documenting the individual extended commands, we will call them *functions* to avoid confusion between them and the one or two character *commands*. We will also use "M-X" as a title like "Mr." for functions, as in "use M-X Foo". The "extended command" is what you *type*, starting with M-X, and what the command *does* is call a function. The name that goes in the command is the name of the command and is also the name of the function, and both terms will be used.

Note: Extended commands and functions were once called "MM commands", but this term is obsolete. If you see it used either in INFO documentation of in Help documentation, please report it. Ordinary one or two character commands used to be known as "^R" commands; please report any occurrences of this obsolete term also.

There are a great many functions in EMACS for you to call. They will be described

5.1.3. Numeric Arguments and String Arguments

Some functions can use numeric prefix arguments. Simply give the Meta-X command an argument and Meta-X will pass it along to the function which it calls. The argument appears before the "M-X" in the prompt, as in "69 M-X", to remind you that the function you call will receive a numeric argument.

Some functions require *string arguments* (sometimes called *suffix arguments*). To specify string arguments, terminate the function name with a single Altmode, then type the arguments, separated by Altmodes. After the last argument, type a Return to cause the function to be executed. For example, the function Describe prints the full documentation of a function (or a variable) whose name must be given as a string argument. An example of using it is

```
Meta-X Describe♦Apropos<cr>
```

which prints the full description of the function named Apropos.

An alternate way of calling extended commands is with the command C-M-X (^R Instant Extended Command). It differs from plain M-X in that the function itself reads any string arguments. The function prompts for each argument individually. If an argument is supposed to be a filename or a command name, completion is available. However, there are compensating disadvantages. For one thing, since the function has already been invoked, you can't rub out from the arguments into the function name. For another, it is not possible to save the whole thing, function name and arguments, for you to recall with C-X Altmode (see below). So C-M-X saves *nothing* for C-X Altmode. The prompt for C-M-X is "C-M-X". You can override it with the variable Instant Command Prompt.

5.1.4. Repeating an Extended Command

The last few extended commands you have executed are saved and you can repeat them. We say that the extended command is saved, rather than that the function is saved, because the whole command, including arguments, is saved.

To re-execute a saved command, use the command C-X Altmode (^R Re-execute Minibuffer). It retypes the last extended command and asks for confirmation. With an argument, it repeats an earlier extended command: 2 means repeat the next to the last command, etc. You can also use the minibuffer to edit a previous extended command and re-execute it with changes (See section 23 [Minibuffer], page 131.).

5.2. Arcane Information about M-X Commands

You can skip this section if you are not interested in customization, unless you want to know what is going on behind the scenes.

5.2.3. Commands and Functions

Actually, *every* command in EMACS simply runs a function. For example, when you type the command C-N, it runs the function "^R Down Real Line". You could just as well do C-U 1 M-X ^R Down Real Line<cr> and get the same effect. C-N can be thought of as a sort of abbreviation. We say that the command C-N has been *connected* to the function ^R Down Real Line. The name is looked up once when the command and function are connected, so that it does not have to be looked up again each time the command is used. For historical reasons, the default argument passed to a function which is connected to a command you typed is 1, but the default for MM and for M-X is 0. This is why the C-U 1 was necessary in the example above. The documentation for individual EMACS commands usually gives the name of the function which really implements the command in parentheses after the command itself.

Just as any function can be called directly with M-X, so almost any function can be connected to a command. You can use the function Set Key to do this. Set Key takes the name of the function as a string argument, then reads the character command (including metizers or other prefix characters) directly from the terminal. To define C-N, you could type

```
M-X Set Key↓^R Down Real Line<cr>
```

and then type C-N. If you use the function View File often, you could connect it to the command C-X V (not normally defined). You could even connect it to the command C-M-V, replacing that command's normal definition. Set Key is good for redefining commands in the middle of editing. An init file or EVARS file can do it each time you run EMACS. See section 22.6 [Init], page 123.

5.2.4. Subroutines

EMACS is composed of a large number of functions, each with a name. Some of these functions are connected to commands; some are there for you to call with M-X; some are called by other functions. The last group are called subroutines. They usually have names starting with "&", as in "& Read Line", the subroutine which reads a line in the echo area. Although most subroutines have such names, any function can be called as a subroutine. Functions like ^R Down Real Line have names starting with ^R because you are not expected to call them directly, either. The purpose of the "&" or "^R" is to get those function names out of the way of command completion in M-X. M-X allows the command name to be abbreviated if the abbreviation is unique, and the commands that you are not interested in might have names that would interfere and make some useful abbreviation cease to be unique. The funny characters at the front of the name prevent this from happening.

5.2.5. Built-in Functions

Not all of the functions in EMACS are written in TECO. A few of the most frequently used single-character commands have definitions written in machine language. These include self-inserting characters, Rubout, C-F, and others. Such functions

Chapter Six

Moving Up And Down Levels

Subsystems and recursive editing levels are two states in which you are temporarily doing something other than editing the visited file as usual. For example, you might be editing a message that you wish to send, or looking at a documentation file with INFO.

6.1. Subsystems

A *subsystem* is an EMACS function which is an interactive program in its own right: it reads commands in a language of its own, and displays the results. You enter a subsystem by typing an EMACS command which invokes it. Once entered, the subsystem runs until a specific command to exit the subsystem is typed. An example of an EMACS subsystem is INFO, the documentation reading program. Others are Backtrace and TDEBUG, used for debugging TECO programs, and RMAIL and BABYL, used for reading and editing mail files.

The commands understood by a subsystem are usually not like EMACS commands, because their purpose is something other than editing text. For example, INFO commands are designed for moving around in a tree-structured documentation file. In EMACS, most commands are Control or Meta characters because printing characters insert themselves. In most subsystems, there is no insertion of text, so non-Control non-Meta characters can be the commands.

While you are inside a subsystem, the mode line usually gives the name of the subsystem (as well as other information supplied by the subsystem, such as the filename and node name in INFO). You can tell that you are inside a subsystem because the mode line does not start with "EMACS", or with an open bracket ("[") which would indicate a recursive editing level. See section 1.1 [Mode Line], page 6.

Because each subsystem implements its own commands, we cannot guarantee anything about them. However, there are conventions for what certain commands ought to do:

C-]	Aborts (exits without finishing up).
Backspace	Scrolls backward, like M-V in EMACS.
Space	Scrolls forward, like C-V in EMACS.
Q	Exits normally.
X	Begins an extended command, like M-X in EMACS.
Help or ?	Prints documentation on the subsystem's commands.

Not all of these necessarily exist in every subsystem, however.

A recursive editing level differs from a subsystem in that the commands are ordinary EMACS commands (though a handful may have been changed slightly), whereas a subsystem defines its own command language.

The text you edit inside a recursive editing level depends on the command which invoked the recursive editing level. It could be a list of options and values, or a list of tab stop settings, syntax table settings, a message to be sent, or any text that you might wish to compose.

Sometimes in a recursive editing level you edit text of the file you are visiting, just as at top level. Why would this be? Usually because a few commands are temporarily changed. For example, Edit Picture in the PICTURE library defines commands good for editing a picture made out of characters, then enters a recursive editing level. When you exit, the special picture-editing commands go away. Until then, the brackets in the mode line serve to remind you that, although the text you are editing is your file, all is not normal. See section 26 [PICTURE], page 151.

In any case, if the mode line says "[...]" you are inside a recursive editing level, and the way to exit (send the message, redefine the options, get rid of the picture-editing commands, etc.) is with the command C-M-C (^R Exit). See section 6.4 [Exiting], page 27. If you change your mind about the command (you don't want to send the message, or change your options, etc.) then you should use the command C-] (Abort Recursive Edit) to get out. See section 24.1 [Aborting], page 133.

Inside recursive editing levels, the help option Help R is defined to print the full documentation of the command which invoked the recursive editing level. The other normal Help options are still available for asking about commands you want to use while inside the recursive edit.

When the text in the mode line is surrounded by parentheses, it means that you are inside a *Minibuffer*. A minibuffer is a special case of the recursive editing level. Like any other, it can be aborted safely with C-]. See section 23 [Minibuffer], page 131.

6.4. Exiting Levels; Exiting EMACS

C-X C-C	Exit from EMACS to the superior job.
C-M-C	Exit from EMACS or from a recursive editing level.

The general EMACS command to exit is C-M-C (^R Exit). This command is used to exit from a recursive editing level back to the top level of EMACS, and to exit from EMACS at top level back to HACTRN. If your keyboard does not have a Meta key, you must type this command by means of a bit prefix character, as C-C C-C or as Altmode C-C. Note carefully the difference between exiting a recursive editing level and aborting it: exiting allows the command which invoked the recursive editing level to finish its job with the text as you have edited it, whereas aborting cancels whatever the command was going to do. See section 24.1 [Aborting], page 133.

We cannot say in general how to exit a subsystem, since each subsystem defines its own command language, but the convention is to use the character "Q".

You can exit from EMACS back to the superior job, usually HACTRN, at any time,

Chapter Seven

Self-Documentation Commands

EMACS provides extensive self-documentation features which revolve around a single character, called the Help character. At any time while using EMACS, you can type the Help character to ask for help. How to type the Help character depends on the terminal you are using, but aside from that the same character always does the trick. If your keyboard has a key labeled Help (above the H), type that key (together with the Top key). Otherwise the way you type the Help character is actually C-_ (Control-Underscore) followed by an H (this is two characters to type, but let's not worry about that). Whatever it is you have to type, to EMACS it is just the Help character. On some terminals just figuring out how to type a Control-Underscore is hard! Typing Underscore and adding the Control key is what logically ought to work, but on some terminals it does not. Sometimes Control-Shift-O works. On VT-100 terminals, typing Control-/ or Control-? sends a Control-_ character.

If you type Help while you are using a subsystem such as INFO, it prints a list of the commands of that subsystem.

If you type Help in the middle of a multi-character command, it often tells you about what sort of thing you should type next. For example, if you type M-X and then Help, it tells you about M-X and how to type the name of the command. If you finish the function name and the Altmode and then type Help, it tells you about the function you have specified so you can know what arguments it needs. If you type C-X and then type Help, it tells you about the C-X commands.

But normally, when it's time for you to start typing a new command, Help offers you several options for asking about what commands there are and what they do. It prompts with "Doc (? for help):" at the bottom of the screen, and you should type a character to say what kind of help you want. You could type Help or "?" at this point to find out what options are available. The ones you are most likely to need are described here.

The most basic Help options are Help C and Help D. You can use them to ask what a particular command does. Help C is for character commands; type the command you want to know about after the Help and the "C" ("C" stands for Character). Thus, Help C M-F or Help C Altmode F tells you about the M-F command. Help D is for asking about functions (extended commands); type the name of the function and a Return. Thus, Help D Lisp Mode<cr> tells you about M-X Lisp Mode. Help D can also tell you the documentation of a variable, if you give it a variable's name instead of a function's name. "D" stands for "Describe", since Help D actually uses the function Describe to do the work.

Chapter Eight

The Mark and the Region

In general, a command which processes an arbitrary part of the buffer must know where to start and where to stop. In EMACS, such commands usually operate on the text between point and *the mark*. This range of text is called *the region*. To specify a region, you set point to one end of it and mark at the other. It doesn't matter which one is set first chronologically, or which one comes earlier in the text. Here are some commands for setting the mark:

C-@	Set the mark where point is.
C-Space	The same.
C-X C-X	Interchange mark and point.
M-@	Set mark after end of next word. This command and the following three do not move point.
C-M-@	Set mark after end of next Lisp s-expression.
C-<	Set mark at beginning of buffer.
C->	Set mark at end of buffer.
M-H	Put region around current paragraph.
C-M-H	Put region around current Lisp defun.
C-X H	Put region around entire buffer.
C-X C-P	Put region around current page.

For example, if you wish to convert part of the buffer to all upper-case, you can use the C-X C-U command, which operates on the text in the region. You can first go to the beginning of the text to be capitalized, put the mark there, move to the end, and then type C-X C-U. Or, you can set the mark at the end of the text, move to the beginning, and then type C-X C-U. C-X C-U runs the function ^R Uppercase Region, whose name signifies that the region, or everything between point and the mark, is to be capitalized.

The most common way to set the mark is with the C-@ command or the C-Space command (^R Set/Pop Mark). They set the mark where point is. Then you can move point away, leaving the mark behind.

It isn't actually possible to type C-Space on non-Meta keyboards. Yet on many terminals the command appears to work anyway! This is because trying to type a Control-Space on those terminals actually sends the character C-@, which means the same thing as C-Space. A few keyboards just send a Space. If you have one of them, you type C-@, or customize your EMACS.

Since terminals have only one cursor, there is no way for EMACS to show you where the mark is located. You have to remember. The usual solution to this problem is to

marks to drift, but they will still be good for this purpose because they are unlikely to drift very far.

Some commands whose primary purpose is to move point a great distance take advantage of the stack of marks to give you a way to undo the command. The best example is M-<, which moves to the beginning of the buffer. It sets the mark first, so that you can use C-U C-@ or C-X C-X to go back to where you were. Searches sometimes set the mark; it depends on how far they move. Because of this uncertainty, searches type out "^@" if they set the mark. The normal situation is that searches leave the mark behind if they move at least 500 characters, but you can change that value since it is kept in the variable Auto Push Point Option. By setting it to 0, you can make all searches set the mark. By setting it to a very large number such as ten million, you can prevent all searches from setting the mark. The string to be typed out when this option does its thing is kept in the variable Auto Push Point Notification.

Chapter Nine

Killing and Moving Text

The commonest way of moving or copying text with EMACS is to kill it, and get it back again in one or more places. This is very safe because the last several pieces of killed text are all remembered, and it is versatile, because the many commands for killing syntactic units can also be used for moving those units. There are also other ways of moving text for special purposes.

9.1. Deletion and Killing

Most commands which erase text from the buffer save it so that you can get it back if you change your mind, or move or copy it to other parts of the buffer. These commands are known as *kill* commands. The rest of the commands that erase text do not save it; they are known as *delete* commands. The delete commands include C-D and Rubout, which delete only one character at a time, and those commands that delete only spaces or line separators. Commands that can destroy significant amounts of nontrivial data generally kill. The commands' names and individual descriptions use the words "kill" and "delete" to say which they do. If you do a kill command by mistake, you can use the Undo command to undo it (See section 24.3 [Undo], page 136.).

C-D	Delete next character.
Rubout	Delete previous character.
M-\	Delete spaces and tabs around point.
C-X C-O	Delete blank lines around the current line.
M-^	Join two lines by deleting the CRLF and any indentation.
C-K	Kill rest of line or one or more lines.
C-W	Kill region (from point to the mark).
M-D	Kill word.
M-Rubout	Kill word backwards.
C-X Rubout	Kill back to beginning of sentence.
M-K	Kill to end of sentence.
C-M-K	Kill s-expression.
C-M-Rubout	Kill s-expression backwards.

9.1.3. Other Kill Commands

A kill command which is very general is C-W (^R Kill Region), which kills everything between point and the mark. With this command, you can kill any contiguous characters, if you first set the mark at one end of them and go to the other end.

Other syntactic units can be killed: words, with M-Rubout and M-D (See section 11.1 [Words], page 45.); s-expressions, with C-M-Rubout and C-M-K (See section 20.6.1 [S-expressions], page 97.); sentences, with C-X Rubout and M-K (See section 11.2 [Sentences], page 46.).

9.2. Un-Killing

Un-killing is getting back text which was killed. The usual way to move or copy text is to kill it and then un-kill it one or more times.

C-Y	Yank (re-insert) last killed text.
M-Y	Replace re-inserted killed text with the previously killed text.
M-W	Save region as last killed text without killing.
C-M-W	Append next kill to last batch of killed text.

Killed text is pushed onto a *ring buffer* called the *kill ring* that remembers the last 8 blocks of text that were killed. (Why it is called a ring buffer will be explained below). The command C-Y (^R Un-kill) reinserts the text of the most recent kill. It leaves the cursor at the end of the text, and puts the mark at the beginning. Thus, a single C-W undoes the C-Y (M-X Undo also does so). C-U C-Y leaves the cursor in front of the text, and the mark after. This is only if the argument is specified with just a C-U, precisely. Any other sort of argument, including C-U and digits, has an effect described below.

If you wish to copy a block of text, you might want to use M-W (^R Copy Region), which copies the region into the kill ring without removing it from the buffer. This is approximately equivalent to C-W followed by C-Y, except that M-W does not mark the buffer as "changed" and does not temporarily change the screen.

There is only one kill ring, and switching buffers or files has no effect on it. After visiting a new file, whatever was last killed in the previous file is still on top of the kill ring. This is important for moving text between files.

9.2.1. Appending Kills

Normally, each kill command pushes a new block onto the kill ring. However, two or more kill commands in a row combine their text into a single entry on the ring, so that a single C-Y command gets it all back as it was before it was killed. This means that you don't have to kill all the text in one command; you can keep killing line after line, or word after word, until you have killed it all, and you can still get it all back at once. (This we join television in leading people to kill thoughtlessly).

Commands that kill forward from point add onto the end of the previous killed text.

9.3. Other Ways of Copying Text

Usually we copy or move text by killing it and un-killing it, but there are other ways that are useful for copying one block of text in many places, or for copying many scattered blocks of text into one place. In addition to those described here, see the self-documentation of the MOVE library.

9.3.1. Accumulating Text

You can accumulate blocks of text from scattered locations either into a buffer or into a file if you like.

To append them into a buffer, use the command C-X A <buffername><cr> (^R Append to Buffer), which inserts a copy of the region into the specified buffer at the location of point in that buffer. If there is no buffer with the name you specify, one is created. If you append text into a buffer which has been used for editing, the copied text goes into the middle of the text of the buffer, wherever point happens to be in it.

Point in that buffer is left at the end of the copied text, so successive uses of C-X A accumulate the text in the specified buffer in the same order as they were copied. If C-X A is given an argument, point in the other buffer is left before the copied text, so successive uses of C-X A add text in reverse order.

You can retrieve the accumulated text from that buffer with M-X Insert Buffer↓<buffername><cr>. This inserts a copy of the text in that buffer into the selected buffer. You can also select the other buffer for editing. See section 14 [Buffers], page 71, for background information on buffers.

Strictly speaking, C-X A does not always append to the text already in the buffer. But if it is used on a buffer which starts out empty, it does keep appending to the end.

Instead of accumulating text within EMACS, in a buffer, you can append text directly into a disk file with the command M-X Append to File↓<filename><cr>. It adds the text of the region to the end of the specified file. M-X Prepend to File adds the text to the beginning of the file instead. The file is changed immediately on disk. These commands are normally used with files that are *not* being visited in EMACS. They have the advantage of working even on files too large to fit into the EMACS address space.

9.3.2. Copying Text Many Times

When you want to insert a copy of the same piece of text frequently, the kill ring becomes impractical, since the text moves down on the ring as you edit, and will be in an unpredictable place on the ring when you need it again. For this case, you can use the commands C-X X (^R Put Q-register) and C-X G (^R Get Q-register) to move the text.

C-X X<q> stores a copy of the text of the region in a place called q-register <q>. With an argument, C-X X deletes the text as well. <q> can be a letter or a digit. This gives 36 places in which you can store a piece of text. C-X G<q> inserts in the buffer

Chapter Ten

Searching

Like other editors, EMACS has commands for searching for an occurrence of a string. The search command is unusual in that it is *incremental*; it begins to search before you have finished typing the search string. As you type in the search string, EMACS shows you where it would be found. When you have typed enough characters to identify the place you want, you can stop. Depending on what you will do next, you may or may not need to terminate the search explicitly with an Altmode first.

C-S	Search forward.
C-R	Search backward.
C-S ♦ C-W	Word search, ignoring whitespace.

The command to search is C-S (^R Incremental Search). C-S reads in characters and positions the cursor at the first occurrence of the characters that you have typed. If you type C-S and then F, the cursor moves right after the first "F". Type an "O", and see the cursor move to after the first "FO". After another "O", the cursor is after the first "FOO" after the place where you started the search. At the same time, the "FOO" has echoed at the bottom of the screen.

If you type a mistaken character, you can rub it out. After the FOO, typing a rubout makes the "O" disappear from the bottom of the screen, leaving only "FO". The cursor moves back to the "FO". Rubbing out the "O" and "F" moves the cursor back to where you started the search.

When you are satisfied with the place you have reached, you can type an Altmode, which stops searching, leaving the cursor where the search brought it. Also, any command not specially meaningful in searches stops the searching and is then executed. Thus, typing C-A would exit the search and then move to the beginning of the line. Altmode is necessary only if the next command you want to type is a printing character, Rubout, Altmode, C-Q, or another search command, since those are the characters that have special meanings inside the search.

Sometimes you search for "FOO" and find it, but not the one you expected to find. There was a second FOO that you forgot about, before the one you were looking for. Then type another C-S and the cursor will find the next FOO. This can be done any number of times. If you overshoot, you can rub out the C-S's.

After you exit a search, you can search for the same string again by typing just C-S C-S: one C-S command to start the search and then another C-S to mean "search again".

If your string is not found at all, the echo area says "Failing I-Search". The cursor

A non-incremental search is also available. Type Altmode (or the Search Exit Char) right after the C-S to get it. Do

```
M-X Describe◆^R String Search<cr>
```

for details. Some people who prefer non-incremental searches put that function on Meta-S, and ^R Character Search (do M-X Describe◆ for details) on C-S. It can do one useful thing which incremental search cannot: search for words regardless of where the line breaks.

Word search searches for a sequence of words without regard to how the words are separated. More precisely, you type a string of many words, using single spaces to separate them, and the string can be found even if there are multiple spaces or line separators between the words. Other punctuation such as commas or periods must match exactly. This is useful in conjunction with documents formatted by text justifiers. If you edit while looking at the printed, formatted version, you can't tell where the line breaks are in the source file. With word search, you can search without having to know.

Word search is a special case of non-incremental search and is invoked with C-S Altmode C-W. This is followed by the search string, which must always be terminated with an Altmode. Searching does not start until the final Altmode is typed. Search Exit Char and Search Exit Option do not apply to word search.

You do not even have to type each word in full, in a word search. An abbreviation is good enough. Word search finds the first occurrence of a sequence of words whose beginnings match the words of the argument.

Chapter Eleven

Commands for English Text

EMACS enables you to manipulate words, sentences, or paragraphs of text. In addition, there are commands to fill text, and convert case. For text-justifier input files, there are commands that may help manipulate font change commands and underlining.

Editing files of text in a human language ought to be done using Text mode rather than Fundamental mode. Invoke M-X Text Mode to enter Text mode. See section 20.1 [Major Modes], page 91. M-X Text Mode causes Tab to run the function ^R Tab to Tab Stop, which allows you to set any tab stops with M-X Edit Tab Stops (See section 11.3 [Indentation], page 48.). Features concerned with comments in programs are turned off except when explicitly invoked. Automatic display of parenthesis matching is turned off, which is what most people want. Finally, the syntax table is changed so that periods are not considered part of a word, while apostrophes, backspaces and underlines are.

If you are editing input for the text justifier TEX, you might want to use TEX mode instead of Text mode. See the file INFO;ETEX >. For editing SCRIBE input, use SCRIBE mode. See section 11.8 [SCRIBE], page 53. Someday there may be special major modes for other text justifiers.

11.1. Word Commands

EMACS has commands for moving over or operating on words. By convention, they are all Meta- characters.

M-F	Move Forward over a word.
M-B	Move Backward over a word.
M-D	Kill up to the end of a word.
M-Rubout	Kill back to the beginning of a word.
M-@	Mark the end of the next word.
M-T	Transpose two words; drag a word forward or backward across other words.

Notice how these commands form a group that parallels the character based commands C-F, C-B, C-D, C-T and Rubout. M-@ is related to C-@.

The commands Meta-F (^R Forward Word) and Meta-B (^R Backward Word) move forward and backward over words. They are thus analogous to Control-F and

M-H Put point and mark around this paragraph (around the following one, if between paragraphs).

C-X Rubout
Kill back to the beginning of the sentence.

11.2.1. Sentences

The commands Meta-A and Meta-E (^R Backward Sentence and ^R Forward Sentence) move to the beginning and end of the current sentence, respectively. They were chosen to resemble Control-A and Control-E, which move to the beginning and end of a line. Unlike them, Meta-A and Meta-E if repeated or given numeric arguments move over successive sentences. EMACS considers a sentence to end wherever there is a ".", "?" or "!" followed by the end of a line or two spaces, with any number of ")"'s, "]"'s, "'"'s, or '"' 's allowed in between. Neither M-A nor M-E moves past the CRLF or spaces which delimit the sentence.

Just as C-A and C-E have a kill command, C-K, to go with them, so M-A and M-E have a corresponding kill command M-K (^R Kill Sentence) which kills from point to the end of the sentence. With minus one as an argument it kills back to the beginning of the sentence. Larger arguments serve as a repeat count.

There is a special command, C-X Rubout (^R Backward Kill Sentence) for killing back to the beginning of a sentence, because this is useful when you change your mind in the middle of composing text.

11.2.2. Paragraphs

Meta-[(^R Backward Paragraph) moves to the beginning of the current or previous paragraph, while Meta-] (^R Forward Paragraph) moves to the end of the current or next paragraph. Blank lines and text justifier command lines separate paragraphs and are not part of any paragraph. Also, an indented line starts a new paragraph.

In major modes for programs (as opposed to Text mode), paragraphs are determined only by blank lines. This makes the paragraph commands continue to be useful even though there are no paragraphs per se.

When there is a fill prefix, then paragraphs are delimited by all lines which don't start with the fill prefix. See section 11.4 [Filling], page 50.

When you wish to operate on a paragraph, you can use the command Meta-H (^R Mark Paragraph) to set the region around it. This command puts point at the beginning and mark at the end of the paragraph point was in. Before setting the new mark at the end, a mark is set at the old location of point; this allows you to undo a mistaken Meta-H with two C-U C-@'s. If point is between paragraphs (in a run of blank lines, or at a boundary), the paragraph following point is surrounded by point and mark. Thus, for example, Meta-H C-W kills the paragraph around or after point.

One way to make an "invisible" paragraph boundary that does not show if the file is printed is to put space-backspace at the front of a line. The space makes the line appear (to the EMACS paragraph commands) to be indented, which usually means that it starts a paragraph.

There are also commands for changing the indentation of several lines at once. Control-Meta-\ (^R Indent Region) gives each line which begins in the region the "usual" indentation by invoking Tab at the beginning of the line. A numeric argument specifies the indentation, and each line is shifted left or right so that it has exactly that much. C-X Tab (^R Indent Rigidly) moves all of the lines in the region right by its argument (left, for negative arguments). The whole group of lines move rigidly sideways, which is how the command gets its name.

11.3.1. Tab Stops

For typing in tables, you can use Text mode's definition of Tab, ^R Tab to Tab Stop, which may be given anywhere in a line, and indents from there to the next tab stop. If you are not in Text mode, this function can be found on M-I anyway.

Set the tab stops using Edit Tab Stops, which allows you to edit some text which defines the tab stops. Here is what it would look like for ordinary tab stops every eight columns.

```
        :       :       :       :       :       :

  123456789 123456789 123456789 123456789 123456789 1234
0         10        20        30        40        50
```

The first line contains a colon or period at each tab stop. Colon indicates an ordinary tab, which fills with whitespace; a period specifies that characters be copied from the corresponding columns of the second line below it. Thus, you can tab to a column automatically inserting dashes or periods, etc. It is your responsibility to put in the second line the text to be copied. In the example above there are no periods, so the second line is not used, and is left blank.

The third and fourth lines you see contain column numbers to help you edit. They are only there while you are editing the tab stops; they are not really part of the tab settings. The first two lines reside in the variable Tab Stop Definitions when they are not being edited. If the second line is not needed, Tab Stop Definitions can be just one line, with no CRLFs. This makes it easier to set the variable in a local modes list. See section 22.7 [Locals], page 127.

EMACS normally uses both tabs and spaces to indent lines, and displays tab characters using eight-character tab stops. (How the ASCII character tab is displayed has nothing to do with the definition of the Tab character as a command). If you prefer, all indentation can be made from spaces only. To request this, turn off Indent Tabs mode with the command M-X Indent Tabs Mode. To display tabs with different tab stops, set the TECO flag FS TAB WIDTH♦ to the desired interval. This is useful for displaying files brought from other operating systems whose normal tab stop spacing is not 8. See section 22.5 [FS Flags], page 121.

To convert all tabs in a file to spaces, you can use M-X Untabify. M-X Tabify performs the opposite transformation, replacing spaces with tabs whenever possible, but only if there are at least three of them so as not to obscure ends of sentences. A numeric argument to Tabify or Untabify specifies the interval between tab stops to use for computing how to change the file. By default, they use the same interval being

The maximum line width for filling is in the variable Fill Column. Both M-Q and Auto Fill make sure that no line exceeds this width. The easiest way to set the variable is to use the command C-X F (^R Set Fill Column), which places the margin at the column point is on, or at the column specified by a numeric argument. The fill column is initially column 70.

To fill a paragraph in which each line starts with a special marker (which might be a few spaces, giving an indented paragraph), use the *fill prefix* feature. Move point to a spot right after the special marker and give the command C-X Period (^R Set Fill Prefix). Then, filling the paragraph will remove the marker from each line beforehand, and put the marker back in on each line afterward. Auto Fill when there is a fill prefix inserts the fill prefix at the front of each new line. Also, any line which does not start with the fill prefix is considered to start a paragraph. To turn off the fill prefix, do C-X Period with point at the front of a line. The fill prefix is kept in the variable Fill Prefix.

The command C-X = (What Cursor Position) can be used to find out the column that the cursor is in, and other miscellaneous information about point which is quick to compute. It prints a line in the echo area that looks like this:

```
X=5 Y=7 CH=101 .=3874(35% of 11014) H=<3051,4640>
```

In this line, the X value is the column the cursor is in (zero at the left), the Y value is the screen line that the cursor is in (zero at the top), the CH value is the octal value of the character after point (101 is "A"), the "point" value is the number of characters in the buffer before point, and the values in parentheses are the percentage of the buffer before point and the total size of the buffer.

The H values are the virtual buffer boundaries, indicate which part of the buffer is still visible when narrowing has been done. If you have not done narrowing, the H values are omitted. For more information about the virtual buffer boundaries, See section 17 [Narrowing], page 81.

11.5. Case Conversion Commands

EMACS has commands for converting either a single word or any arbitrary range of text to upper case or to lower case.

M-L	Convert following word to lower case.
M-U	Convert following word to upper case.
M-C	Capitalize the following word.
C-X C-L	Convert region to lower case.
C-X C-U	Convert region to upper case.

The word conversion commands are the most useful. Meta-L (^R Lowercase Word) converts the word after point to lower case, moving past it. Thus, successive Meta-L's convert successive words. Meta-U (^R Uppercase Word) converts to all capitals instead, while Meta-C (^R Uppercase Initial) puts the first letter of the word into upper case and the rest into lower case. All these commands convert several words at once if given an argument. They are especially convenient for converting a large amount of text from all upper case to mixed case, because you can move through the text using M-L, M-U or M-C on each word as appropriate.

11.7. Underlining

EMACS has two commands for manipulating text-justifier underlining command characters. These commands do not produce any sort of overprinting in the text file itself; they insert or move command characters which direct text justifiers to produce underlining. By default, commands for the text justifier R are used.

M-_	Underline previous word or next word.
C-X _	Underline region.

M-_ is somewhat like M-# in that it either creates an underline around the previous word or extends it past the next word. However, where a font change requires that you specify a font number, an underline is just an underline and has no parameter for you to specify. Also, it is assumed that the text justifier's commands for starting and ending underlines are distinguishable, whereas you can't tell from a font change whether it is "starting" something or "ending" something. M-_ differs slightly from M-# as a result.

M-_ with no argument creates an underline around the previous word if there is none. If there is an underline there, it is extended one word forward. Thus, you can insert an underlined word by typing the word and then a M-_. Or you can underline several existing words by moving past the first of them, and typing one M-_ for each word.

M-_ given in the vicinity of an underline-begin moves *it* forward. Thus, it should be thought of as applying to any boundary, where underlining either starts or stops, and moving it forward. If a begin underlining is moved past an end, or vice versa, they both disappear.

Giving M-_ an argument merely tells it to apply to several words at once instead of one. M-_ with a positive argument of n underlines the next n words, either creating an underlined area or extending an existing one. With a negative argument, that many previous words are underlined. Thus, M-_ can do more things with underlines than M-# can do with font changes, because of the facts that you don't need to use the argument to say which font, and you can tell a beginning from an end.

For larger scale operations, you can use C-X _ to place underlines from point to mark, or C-X _ with a negative argument to remove all underlining between point and mark.

By default, ↑B is used to begin an underline and ↑E is used to end one. The variables Underline Begin and Underline End may be created and set to strings to use instead. For a single character you can use the numeric ASCII code for it.

11.8. SCRIBE Mode

SCRIBE mode provides many special editing commands for manipulating the commands for the text justifier SCRIBE. Instances of SCRIBE commands are referred to in EMACS as *environments*, though strictly speaking it is the command name which is the environment, and not all commands either.

starting with page delimiters are paragraph delimiters as usual, and lines starting with space or tab start paragraphs as usual.

The comment commands know that comments in SCRIBE files start with "@Comment{" and end with "}". See section 20.5 [Comments], page 95.

You can use the M-X Compile command to invoke SCRIBE. See section 20.2 [Compile], page 92. It operates on the file currently visited. You can use a string argument to specify switches. After SCRIBE is finished, you must type ♦P to resume EMACS. Then, if there were any errors, EMACS splits the screen and displays the errors in the bottom window. The command M-X Next Scribe Error moves to the point in the file at which the next error occurred. Currently the only ITS machine on which SCRIBE is installed is MC.

The functions listed in this section live in the library SCRIBE, which is loaded automatically if you enter SCRIBE mode.

11.9. Dissociated Press

M-X Dissociated Press is a command for scrambling a file of text either word by word or character by character. Starting from a bufferfull of straight English, it produces extremely amusing output. Dissociated Press prints its output on the terminal. It does not change the contents of the buffer.

Dissociated Press operates by jumping at random from one point in the buffer to another. In order to produce plausible output rather than gibberish, it insists on a certain amount of overlap between the end of one run of consecutive words or characters and the start of the next. That is, if it has just printed out "president" and then decides to jump to a different point in the file, it might spot the "ent" in "pentagon" and continue from there, producing "presidentagon". Long sample texts produce the best results.

A negative argument to M-X Dissociated Press tells it to operate character by character, and specifies the number of overlap characters. A positive argument tells it to operate word by word and specifies the number of overlap words. In this mode, whole words are treated as the elements to be permuted, rather than characters. No argument is equivalent to an argument of two. For your againformation, the output is only printed on the terminal. The file you start with is not changed.

Dissociated Press produces nearly the same results as a Markov chain based on a frequency table constructed from the sample text. It is, however, an independent, ignoriginal invention. Dissociated Press techniquitously copies several consecutive characters from the sample between random choices, whereas a Markov chain would choose randomly for each word or character. This makes for more plausible sounding results.

It is a mustatement that too much use of Dissociated Press can be a developediment to your real work. Sometimes to the point of outragedy. And keep dissociwords out of your documentation, if you want it to be well usorenced and properbose. Have fun. Your buggestions are welcome.

Chapter Twelve

Commands for Fixing Typos

In this section we describe the commands that are especially useful for the times when you catch a mistake in your text just after you have made it, or change your mind while composing text on line.

Rubout	Delete last character.
M-Rubout	Kill last word.
C-X Rubout	Kill to beginning of sentence.
C-T	Transpose two characters.
C-X C-T	Transpose two lines.
C-X T	Transpose two arbitrary regions.
M-Minus M-L	Convert last word to lower case.
M-Minus M-U	Convert last word to all upper case.
M-Minus M-C	Convert last word to lower case with capital initial.
M-'	Fix up omitted shift key on digit.
M-$	(Meta-Dollar Sign) Check and correct spelling of word.
Correct Spelling	Check and correct spelling of entire buffer.

12.1. Killing Your Mistakes

The Rubout command is the most important correction command. When used among printing (self-inserting) characters, it can be thought of as canceling the last character typed.

When your mistake is longer than a couple of characters, it might be more convenient to use M-Rubout or C-X Rubout. M-Rubout kills back to the start of the last word, and C-X Rubout kills back to the start of the last sentence. C-X Rubout is particularly useful when you are thinking of what to write as you type it, in case you change your mind about phrasing. M-Rubout and C-X Rubout save the killed text for C-Y and M-Y to retrieve (See section 9.2 [Un-killing], page 37.).

M-Rubout is often useful even when you have typed only a few characters wrong, if you know you are confused in your typing and aren't sure exactly what you typed. At such a time, you cannot correct with Rubout except by looking at the screen to see what you did. It requires less thought to kill the whole word and start over again, especially if the system is heavily loaded.

12.4. Checking and Correcting Spelling

When you write a paper, you should correct its spelling at some point close to finishing it (and maybe earlier as well). To correct the entire buffer, do M-X Correct Spelling. This invokes the SPELL spelling corrector program, which will ask you what to do with each misspelled word. Refer to its documentation. When it finished, you will be back in EMACS.

To check the spelling of the word before point, and optionally correct it as well, use the command M-$ (^R Correct Word Spelling). This is a Dollar sign, not an Altmode! This command sends the word to the SPELL program for correction.

If SPELL recognizes the word as a correctly spelled one (although not necessarily the one you meant!) you will see "Found it" or "Found it because of" followed by the word without its suffix. If the program cannot at all recognize the word, it will print "Couldn't find it."

If SPELL recognizes the word as a misspelling, it displays on the screen the other words which are possibilities for the correct spelling, and gives each one a number. Then, you can type one of the following things:

0 to 9	Replace misspelled word with that spelling (preserving case, just like Replace String and Query Replace, unless Case Replace is zero).
Space	Exit and make no changes.
%	Read a digit as above and Query Replace (M-%) the incorrect spelling with the correct one from the beginning of the buffer.

No other responses are allowed.

The cursor need not be immediately after the word you want to correct; it can be in the middle, or following any word-separator characters after the end of the word. Note that the major mode you are using affects which characters are word-separators. See section 22.4 [Syntax Table], page 119.

The first time you use an EMACS spelling correction command, it creates a SPELL job for you. From then on it uses the same SPELL job. It's ok to kill the job if you don't think you'll be using M-$ again during that session. But if you do leave the job around checking words will much quicker. Giving M-$ a negative argument (as in M-- M-$) kills the SPELL job. Of course, you can kill it from DDT also.

If you are a regular user of the SPELL program, you might have a dictionary file of words which you use but which are foreign to SPELL. If there are words in this file which you might want to use M-$ to correct, you can specify that you want this dictionary to be loaded into the spell job which EMACS uses. To do this, set the variable Spell Initialization to the string of SPELL program commands you want to use. For loading a dictionary, this string would be a "Load " followed by the dictionary filename. Other initialization commands for SPELL can appear there also. The commands must be separated by altmodes.

You can also pass arbitrary commands to the SPELL job with command M-X Command to Spell.

Chapter Thirteen

File Handling

The basic unit of stored data is the file. Each program, each paper, lives usually in its own file. To edit a program or paper, the editor must be told the name of the file that contains it. This is called *visiting* the file. To make your changes to the file permanent on disk, you must *save* the file. EMACS also has facilities for deleting files conveniently, and for listing your file directory. Special text in a file can specify the modes to be used when editing the file.

13.1. Visiting Files

C-X C-V	Visit a file.
C-X C-Q	Change regular visiting to read only, or vice versa.
C-X C-S	Save the visited file.
Meta-~	Tell EMACS to forget that the buffer has been changed.

Visiting a file means copying its contents into EMACS where you can edit them. EMACS remembers the name of the file you visited. Unless you use the multiple buffer or window features of EMACS, you can only be visiting one file at a time. The name of the file you are visiting in the currently selected buffer is visible in the mode line when you are at top level, followed by its version number if it has one in parentheses.

The changes you make with EMACS are made in a copy inside EMACS. The file itself is not changed. The changed text is not permanent until you *save* it in a file. The first time you change the text, a star appears at the end of the mode line; this indicates that the text contains fresh changes which will be lost unless you save them.

To visit a file, use the command C-X C-V (^R Visit File). Follow the command with the name of the file you wish to visit, terminated by a Return. If you can see a filename in the mode line, then that name is the default, and any component of the filename which you don't specify is taken from it. If EMACS thinks you can't see the defaults, they are included in the prompt. You can abort the command by typing C-G, or edit the filename with Rubout or C-U. If you do type a Return to finish the command, the new file's text appears on the screen, and its name and version appear in the mode line.

When you wish to save the file and make your changes permanent, type C-X C-S (^R Save File). After the save is finished, C-X C-S prints "Written: <filenames>" in the echo area at the bottom of the screen. If there are no changes to save (no star at the end of the mode line), the file is not saved; it would be redundant to save a duplicate of the previous version.

If EMACS is about to save a file and sees that the date of the latest version on disk does not match what EMACS last read or wrote, EMACS notifies you of this fact, and asks what to do, because this probably means that something is wrong. For example, someone else may have been editing the same file. If this is so, there is a good chance that your work or his work will be lost if you don't take the proper steps. You should first find out exactly what is going on. The C-X C-D command to list the directory will help. If you determine that someone else has modified the file, save your file under different names (or at least making a new version) and then SRCCOM the two files to merge the two sets of changes. Also get in touch with the other person so that he doesn't continue editing.

13.2. How to Undo Drastic Changes to a File

If you have made extensive changes to a file and then change your mind about them, you can get rid of them by reading in the previous version of the file. To do this, use M-X Revert File. If you have been using Auto Save mode, it reads in the last version of the visited file or the last auto save file, whichever is more recent.

In Auto Save mode, saving under special Auto Save filenames, then you can ask to revert to the last "real" save, ignoring subsequent auto saves, with C-U M-X Revert File. If you are using the style of auto saving which saves under the real filenames, this is not possible.

M-X Revert File does not change point, so that if the file was only edited slightly, you will be at approximately the same piece of text after the Revert as before. If you have made drastic changes, the same value of point in the old file may address a totally different piece of text.

Because M-X Revert File can be a disaster if done by mistake, it asks for confirmation (Y or N) before doing its work. A pre-comma argument can be used to inhibit the request for confirmation when you call the function Revert File from a TECO program, as in 1,M(M.M Revert File♦).

13.3. Auto Save Mode: Protection Against Disasters

In Auto Save mode, EMACS saves your file from time to time (based on counting your commands) without being asked. Your file is also saved if you stop typing for more than a few minutes when there are changes in the buffer. This prevents you from losing more than a limited amount of work in a disaster. (Another method of protection is the journal file. See section 24.4 [Journals], page 137.).

You can turn auto saving on or off in an individual buffer with M-X Auto Save. In addition, you can have auto saving by default in all buffers by setting the option Auto Save Default. The frequency of saving, and the number of saved versions to keep, can both be specified.

Each time you visit a file, no matter how, auto saving is turned on for that file if Auto Save Default is nonzero. Once you have visited a file, you can turn auto saving on or

which knows how to find the most recent save, permanent or not under whatever filenames. See section 13.2 [Revert], page 63.

For your protection, if a file has shrunk by more than 30% since the last save, auto saving does not save. Instead it prints a message that the file has shrunk. You can save explicitly if you wish; after that, auto saving will resume.

Although auto saving generates large numbers of files, it does not clog directories, because it cleans up after itself. Only the last Auto Save Max auto save files are kept; as further saves are done, old auto saves are deleted. However, only files made by auto saving (or by explicitly requested auto-saves with C-U C-X C-S) are deleted in this way. If Auto Save Max is 1, then repeated auto saves rewrite the same version of the file; the version number is only incremented after a real save. (It will write a new version if it is unable to rewrite the old one).

The variable Auto Save Max is initially 2. Changing the value may not take effect in a given buffer until you turn auto saving off and on in that buffer.

The number of characters of input between auto saves is controlled by the variable Auto Save Interval. It is initially 500. Changing this takes effect immediately.

If you use the multiple-buffer features of EMACS (See section 14 [Buffers], page 71.) then you may want to have auto saving for all buffers, not just the one that is selected at the moment. To get this, set the variable Auto Save All Buffers nonzero.

13.4. Listing a File Directory

To look at a part of a file directory, use the C-X C-D command (^R Directory Display). With no argument, it shows you the file you are visiting, and related files with the same first name. C-U C-X C-D reads a filename from the terminal and shows you the files related to that filename.

To see the whole directory in a brief format, use the function List Files, which takes the directory name as a string argument. The function View Directory prints a verbose listing of a whole directory.

The variable Auto Directory Display can be set to make many file operations display the directory automatically. The variable is normally 0; making it positive causes write operations such as Write File to display the directory, and making it negative causes read operations such as Insert File or visiting to display it as well. The display is done using the default directory listing function which is kept in the variable Directory Lister. Normally this is the function & Subset Directory that displays only the files related to the current default file. An alternative type of directory listing can be obtained by setting Directory Lister to the function & Rotated Directory Listing. After this function display the particular file you are interested in, it offers to display the rest. If you say "yes" (with a Space), it displays through the end of the directory, and around to the beginning of the directory, stopping where it originally started. See section 22.6 [Init], page 123.

The function List Directories prints an alphabetized list of all the file directories on the system.

13.6.1. Basic DIRED Commands

You can mark a file for deletion by moving to the line describing the file and typing D, C-D, K, or C-K. The deletion mark is visible as a D at the beginning of the line. Point is moved to the beginning of the next line, so that several D's delete several files. Alternatively, if you give D an argument it marks that many consecutive files. Given a negative argument, it marks the preceding file (or several files) and puts point at the first (in the buffer) line marked. Most of the DIRED commands (D, U, I, $, P, S, C, E, Space) repeat this way with numeric arguments.

If you wish to remove a deletion mark, use the U (for Undelete) command, which is invoked like D; it removes the deletion mark from the current line (or next few lines, if given an argument). The Rubout command removes the deletion mark from the previous line, moving up to that line. Thus, a Rubout after a D precisely cancels the D.

For extra convenience, Space is made a command similar to C-N. Moving down a line is done so often in DIRED that it deserves to be easy to type. Rubout is often useful simply for moving up.

If you are not sure whether you want to delete a file, you can examine it by typing E. This enters a recursive editing mode on the file, which you can exit with C-M-C. The file is not really visited at that time, and you are not allowed to change it. When you exit the recursive editing level, you return to DIRED. The V command is like E but uses View File to look at the file.

When you have marked the files you wish to mark, you can exit DIRED with C-M-C. If any files were marked for deletion, DIRED lists them in a concise format, several per line. A file with "!" appearing next to it in this list has not been saved on tape and will be gone forever if deleted. A file with ">" in front of it is the most recent version of a sequence and you should be wary of deleting it. Then DIRED asks for confirmation of the list. You can type "YES" (Just "Y" won't do) to go ahead and delete them, "N" to return to editing the directory so you can change the marks, or "X" to give up and delete nothing. No Return character is needed. No other inputs are accepted at this point.

13.6.2. Other DIRED Commands

The "!" command moves down (or up, with an argument of -1) to the next undumped file (one with a "!" before its date).

N finds the next "hog": the next file which has at least three versions (or, more than File Versions Kept).

T when given on a line describing a link marks for deletion the file which the link points to. This file need not be in the directory you are editing to be deleted in this way.

1 copies the file you are pointing at to the primary pack. 2 copies it to SECOND:. 3 copies it to THIRD:. 4 copies it to FOURTH:.

$ complements the don't-reap attribute of the file: this is displayed as a dollar sign to the left of the file date.

next screenful or a Backspace to see the previous screenful. Typing anything else exits the command. View File does not visit the file; it does not alter the contents of any buffer. The advantage of View File is that the whole file does not need to be loaded before you can begin reading it. The inability to do anything but page forward or backward is a consequence.

M-X Write File♦<file><cr> writes the contents of the buffer into the file <file>, and then visits that file. It can be thought of as a way of "changing the name" of the file you are visiting. Unlike C-X C-S, Write File saves even if the buffer has not been changed. C-X C-W is another way of getting at this command.

M-X Insert File♦<file><cr> inserts the contents of <file> into the buffer at point, leaving point unchanged before the contents and mark after them. The current defaults are used for <file>, and are updated.

M-X Write Region♦<file><cr> writes the region (the text between point and mark) to the specified file. It does not set the visited filenames. The buffer is not changed.

M-X Append to File♦<file><cr> appends the region to <file>. The text is added to the end of <file>.

M-X Prepend to File♦<file><cr> adds the text to the beginning of <file> instead of the end.

M-X Set Visited Filename♦<file><cr> changes the name of the file being visited without reading or writing the data in the buffer. M-X Write File is approximately equivalent to this command followed by a C-X C-S.

M-X Delete File♦<file><cr> deletes the file. If you did not get a chance to see the default filenames before typing the filename to delete, then Delete File asks for confirmation before deleting the file.

M-X Copy File♦<old file>♦<new file><cr> copies the file.

M-X Rename File♦<old name>♦<new name><cr> renames the file.

The default filenames for all of these operations are *TECO default filenames*. Most of these operations also leave the TECO default names set to the file they operated on. The TECO default is *not always* the same as the file you are visiting. When you visit a file, they start out the same; the commands mentioned above change the TECO default, but do not change the visited filenames. Each buffer has its own TECO default filenames.

The operation of visiting a file is available as a function under the name M-X Visit File♦<file><cr>. In this form, it uses the TECO default as its defaults, though it still sets both the TECO default and the visited filenames.

13.8. The Directory Comparison Subsystem

The function Compare Directories makes it easy to compare two directories to see which files are present in both and which are present only in one. It compares a directory on the local machine with the directory of the same name on another machine.

Chapter Fourteen

Using Multiple Buffers

When we speak of "the buffer", which contains the text you are editing, we have given the impression that there is only one. In fact, there may be many of them, each with its own body of text. At any time only one buffer can be *selected* and available for editing, but it isn't hard to switch to a different one. Each buffer individually remembers which file it is visiting, what modes are in effect, and whether there are any changes that need saving.

C-X B	Select or create a buffer.
C-X C-F	Visit a file in its own buffer.
C-X C-B	List the existing buffers.
C-X K	Kill a buffer.

Each buffer in EMACS has a single name, which normally doesn't change. A buffer's name can be any length. The name of the currently selected buffer, and the name of the file visited in it, are visible in the mode line when you are at top level. A newly started EMACS has only one buffer, named "Main".

As well as the visited file and the major mode, a buffer can, if ordered to, remember many other things *locally*, which means, independently of all other buffers. See section 22.3 [Variables], page 118.

14.1. Creating and Selecting Buffers

To create a new buffer, you need only think of a name for it (say, "FOO") and then do C-X B FOO<cr>, which is the command C-X B (Select Buffer) followed by the name. This makes a new, empty buffer and selects it for editing. The new buffer is not visiting any file, so if you try to save it you will be asked for the filenames to use. Each buffer has its own major mode; the new buffer's major mode is taken from the value of the variable Default Major Mode, or from the major mode of the previously selected buffer if the value of Default Major Mode is the null string. Normally the Default Major Mode is Fundamental mode.

To return to buffer FOO later after having switched to another, the same command C-X B FOO<cr> is used, since C-X B can tell whether a buffer named FOO exists already or not. It does not matter whether you use upper case or lower case in typing the name of a buffer. C-X B Main<cr> reselects the buffer Main that EMACS started out with. Just C-X B<cr> reselects the previous buffer. Repeated C-X B<cr>'s alternate between the last two buffers selected.

The commands C-X A (^R Append to Buffer) and M-X Insert Buffer can be used to copy text from one buffer to another. See section 9.3 [Copying], page 39.

14.3. Killing Buffers

After you use an EMACS for a while, it may fill up with buffers which you no longer need. Eventually you can reach a point where trying to create any more results in an "URK" error. So whenever it is convenient you should do M-X Kill Some Buffers, which asks about each buffer individually. You can say Y or N to kill it or not. Or you can say Control-R to take a look at it first. This does not actually select the buffer, as the mode line shows, but gives you a recursive editing level in which you can move around and look at things. When you have seen enough to make up your mind, exit the recursive editing level with a C-M-C and you will be asked the question again. If you say to kill a buffer that needs saving, you will be asked whether it should be saved. See section 24.2.5 [Storage Exhausted], page 135. .

You can kill the buffer FOO by doing C-X K FOO<cr>. You can kill the selected buffer, a common thing to do if you use C-X C-F, by doing C-X K<cr>. If you kill the selected buffer, in any way, EMACS asks you which buffer to select instead. Saying just <cr> at that point tells EMACS to choose one reasonably. C-X K runs the function Kill Buffer.

Chapter Fifteen

Controlling the Display

Since only part of a large file fits on the screen, EMACS tries to show the part that is likely to be interesting. The display control commands allow you to ask to see a different part of the file.

C-L	Clear and redisplay screen, putting point at a specified vertical position.
C-V	Scroll forwards (a screen or a few lines).
M-V	Scroll backwards.
M-R	Move point to the text at a given vertical position.
C-M-R	Shift the function point is in onto the screen.

The terminal screen is rarely large enough to display all of your file. If the whole buffer doesn't fit on the screen, EMACS shows a contiguous portion of it, containing point. It continues to show approximately the same portion until point moves outside of it; then EMACS chooses a new portion centered around the new point. This is EMACS's guess as to what you are most interested in seeing. But if the guess is wrong, you can use the display control commands to see a different portion. The finite area of screen through which you can see part of the buffer is called *the window*, and the choice of where in the buffer to start displaying is also called *the window*.

First we describe how EMACS chooses a new window position on its own. The goal is usually to place point 35 percent of the way down the screen. This is controlled by the variable Cursor Centering Point, whose value is the percent of the screen down from the top. However, if the end of the buffer is on the screen, EMACS tries to leave at most 35 percent of the screen blank beneath it, so that the screen is not wasted. This percentage is controlled by the variable End of Buffer Display Margin. These variables work by controlling FS flags, and their values must never be negative or greater than 99.

Normally EMACS only chooses a new window position if you move point off the screen. However, you can ask for a new window position to be computed whenever point gets too close to the top of the screen by setting the variable Top Display Margin to the percentage of the screen in which point must not appear. Bottom Display Margin does the same thing for a region near the bottom.

The basic display control command is C-L (^R New Window). In its simplest form, with no argument, it clears the screen and tells EMACS to choose a new window position, centering point 35 percent of the way from the top as usual.

C-L with a positive argument chooses a new window so as to put point that many

Chapter Sixteen

Two Window Mode

EMACS allows you to split the screen into two *windows* and use them to display parts of two files, or two parts of the same file.

C-X 2	Start showing two windows.
C-X 3	Show two windows but stay "in" the top one.
C-X 1	Show only one window again.
C-X O	Switch to the Other window
C-X 4	Find buffer, file or tag in other window.
C-X ^	Make this window bigger.
C-M-V	Scroll the other window.

In *two window* mode, the text display portion of the screen is divided into two parts called *windows*, which display different pieces of text. The two windows can display two different files, or two parts of the same file. Only one of the windows is selected; that is the window which the cursor is in. Editing normally takes place in that window alone. To edit in the other window, you would give a special command to move the cursor to the other window, and then edit there. Since there is only one mode line, it applies to the window you are in at the moment.

The command C-X 2 (^R Two Windows) enters two-window mode. A line of dashes appears across the middle of the screen, dividing the text display area into two halves. Window one, containing the same text as previously occupied the whole screen, fills the top half, while window two fills the bottom half. The cursor moves to window two. If this is your first entry to two-window mode, window two contains a new buffer named W2. Otherwise, it contains the same text it held the last time you looked at it.

To return to viewing only one window, use the command C-X 1 (^R One Window). Window one expands to fill the whole screen, and window two disappears until the next C-X 2. C-U C-X 1 gets rid of window one and makes window two use the whole screen. Neither of these depends on which window the cursor is in when the command is given.

While you are in two window mode you can use C-X O (^R Other Window) to switch between the windows. After doing C-X 2, the cursor is in window two. Doing C-X O moves the cursor back to window one, to exactly where it was before the C-X 2. The difference between this and doing C-X 1 is that C-X O leaves window two visible on the screen. A second C-X O moves the cursor back into window two, to where it was before the first C-X O. And so on...

Often you will be editing one window while using the other just for reference. Then,

continues to show the same text. Then, having found in window two the place you wish to refer to, you can go back to window one with C-X O to make your changes. Finally you can do C-X 1 to make window two leave the screen. If you are already in two window mode, C-U C-X O switches windows carrying the buffer from the old window to the new one so that both windows show that buffer.

If you have the same buffer in both windows, you must beware of trying to visit a different file in one of the windows with C-X C-V, because if you bring a new file into this buffer, it will replace the old file in *both* windows. To view different files in the two windows again, you must switch buffers in one of the windows first (with C-X B or C-X C-F, perhaps).

A convenient "combination" command for viewing something in the other window is C-X 4 (^R Visit in Other Window). With this command you can ask to see any specified buffer, file or tag in the other window. Follow the C-X 4 with either B and a buffer name, F or C-F and a file name, or T or "." and a tag name (See section 21 [TAGS], page 107.). This switches to the other window and finds there what you specified. If you were previously in one-window mode, two-window mode is entered. C-X 4 B is similar to to C-X 2 C-X B. C-X 4 F is similar to C-X 2 C-X C-F. C-X 4 T is similar to C-X 2 M-Period. The difference is one of efficiency, and also that C-X 4 works equally well if you are already using two windows.

Chapter Seventeen

Narrowing

Narrowing means focusing in on some portion of the buffer, making the rest temporarily invisible and inaccessible.

C-X N	Narrow down to between point and mark.
C-X P	Narrow down to the page point is in.
C-X W	Widen to view the entire buffer.

When you have narrowed down to a part of the buffer, that part appears to be all there is. You can't see the rest, you can't move into it (motion commands won't go outside the visible part), you can't change it in any way. However, it is not gone, and if you save the file all the invisible text will be saved. In addition to sometimes making it easier to concentrate on a single subroutine or paragraph by eliminating clutter, narrowing can be used to restrict the range of operation of a replace command. The word "Narrow" appears in the mode line whenever narrowing is in effect.

The primary narrowing command is C-X N (^R Narrow Bounds to Region). It sets the *virtual buffer boundaries* at point and the mark, so that only what was between them remains visible. Point and mark do not change.

The way to undo narrowing is to widen with C-X W (^R Widen Bounds). This makes all text in the buffer accessible again.

Another way to narrow is to narrow to just one page, with C-X P (^R Narrow Bounds to Page). See section 18 [Pages], page 83.

You can get information on what part of the buffer you are narrowed down to using the C-X = command. See section 11.4 [Filling], page 50.

The virtual buffer boundaries are a powerful TECO mechanism used internally in EMACS in many ways. While only the commands described here set them so as you can see, many others set them temporarily using the TECO commands FS VB♦ and FS VZ♦, but restore them before they are finished.

Chapter Eighteen

Commands for Manipulating Pages

Files are often thought of as divided into *pages* by the ASCII character formfeed (↑L). For example, if a file is printed on a line printer, each page of the file, in this sense, will start on a new page of paper. Most editors make the division of a file into pages extremely important. For example, they may be unable to show more than one page of the file at any time. EMACS treats a formfeed character just like any other character. It can be inserted with C-Q C-L (or, C-M-L), and deleted with Rubout. Thus, you are free to paginate your file, or not. However, since pages are often meaningful divisions of the file, commands are provided to move over them and operate on them.

C-M-L	Insert formfeed.
C-X C-P	Put point and mark around this page (or another page).
C-X [	Move point to previous page boundary.
C-X]	Move point to next page boundary.
C-X P	Narrow down to just this (or next) page.
C-X L	Count the lines in this page.
M-X What Page	Print current page and line number.

The C-X [(^R Previous Page) command moves point to the previous page delimiter (actually, to right after it). If point starts out right after a page delimiter, it skips that one and stops at the previous one. A numeric argument serves as a repeat count. The C-X] (^R Next Page) command moves forward past the next page delimiter.

The command M-X What Page prints the page and line number of the cursor in the echo area. There is a separate command to print this information because it is likely to be slow and should not slow down anything else. The design of TECO is such that it is not possible to know the absolute number of the page you are in, except by scanning through the whole file counting pages.

The C-X C-P command (^R Mark Page) puts point at the beginning of the current page and the mark at the end. The page delimiter at the end is included (the mark follows it). The page delimiter at the front is excluded (point follows it). This command can be followed by a C-W to kill a page which is to be moved elsewhere. If it is inserted after a page delimiter, at a place where C-X] or C-X [would take you, then the page will be properly delimited before and after once again.

A numeric argument to C-X C-P is used to specify which page to go to, relative to the current one. Zero means the current page. One means the next page, and -1 means the previous one.

be done by giving the page number as an argument to C-X C-P (^R Goto Page). If you give a number too big, the last page in the file is selected.

For convenience, C-X C-P with no argument when you are looking at the whole file selects the page containing point. When you are looking at only one page, C-X C-P with no argument goes to the next page and with a negative argument goes to the previous page.

However, the main commands for moving forward or backward by pages are C-X [and C-X] (^R Goto Previous Page and ^R Goto Next Page). These take a numeric argument (either sign) and move that many pages.

To go back to viewing the whole file instead of just one page, you can use the C-X W (^R PAGE Widen Bounds) command. These are the same characters that you would use in standard EMACS, but they run a different function that knows to remove the page number from the mode line.

The C-S (^R Incremental Search) and C-R (^R Reverse Search) commands are redefined to widen bounds first and narrow them again afterwards. So you can search through the whole file, but afterward see only the page in which the search ended. In fact, PAGE goes through some trouble to work with whatever search functions you prefer to use, and find them wherever you put them.

To split an existing page, you could insert a ↑L, but unless you do this while viewing the whole file, PAGE might get confused. The clean way is to use C-X P (^R Insert Pagemark) which inserts the page mark, and narrows down to the second of the two pages formed from the old page. The clean way to get rid of a page mark is to use C-X J (^R Join Next Page). It gets rid of the page mark after the current page; or, with a negative argument, gets rid of the page mark before this page.

A page mark is defined as <CRLF>↑L. If you set the variable PAGE Flush CRLF to 1, a page mark is <CRLF>↑L<CRLF>, which has the effect of making the CRLF at the beginning of each page invisible. This may be desirable for EMACS library source files. You can also specify some other string in place of ↑L by setting the variable Page Delimiter. If Page Delimiter specifies multiple alternatives, separated by ↑O, PAGE always inserts the first of them, but recognizes them all.

To see a list of all the pages in the file, each one represented by its first nonempty line, use M-X View Page Directory. It prints out the first non-blank line on each page, preceded by its page number. M-X Insert Page Directory inserts the same directory into the buffer at point. If you give it an argument, it tries to make the whole thing into a comment by putting the Comment Start string at the front of each line and the Comment End string at the end.

If the variable Page Setup Hook exists, PAGE will execute its value as the function for placing PAGE's functions on keys. This is done instead of the normal assignments to C-X [, C-X], C-X C-P, C-X P, and C-X J.

Chapter Nineteen

Replacement Commands

Global search-and-replace operations are not needed as often in EMACS as they are in other editors, but they are available. In addition to the simple Replace operation which is like that found in most editors, there is a Query Replace operation which asks you, for each occurrence of the pattern, whether to replace it.

To replace every instance of FOO after point with BAR, you can do

```
M-X Replace◆FOO◆BAR<cr>
```

Replacement occurs only after point, so if you want to cover the whole buffer you must go to the beginning first. Replacement continues to the end of the buffer, but you can restrict it by narrowing. See section 17 [Narrowing], page 81.

Unless the variable Case Replace is zero, Replace tries to preserve case; give both FOO and BAR in lower case, and if a particular FOO is found with a capital initial or all capitalized, the BAR which replaces it will be given the same case pattern. Thus,

```
M-X Replace◆foo◆bar<cr>
```

would replace "foo" with "bar", "Foo" with "Bar" and "FOO" with "BAR". If Case Replace is zero, the replacement string is inserted with the case you used when you typed it. If Case Search is zero, the string to be replaced is found only when it has the same case as what you typed.

If you give Replace (or Query Replace) an argument, then it replaces only occurrences of FOO delimited by break characters (or an end of the buffer). So you can replace only FOO the word, and not FOO when it is part of FOOBAR.

The string FOO to be replaced is actually a TECO search string, a type of pattern, in which the characters ↑B, ↑N, ↑O, ↑Q, ↑S, ↑X, and ↑] are special. See section 19.3 [TECO search strings], page 89.

19.1. Query Replace

If you want to change only some of the occurrences of FOO, not all, then you cannot use an ordinary Replace. Instead, use M-X Query Replace◆FOO◆BAR<cr>. This displays each occurrence of FOO and waits for you to say whether to replace it with a BAR. The things you can type when you are shown an occurrence of FOO are:

Space	to replace the FOO (preserving case, just like plain Replace, unless Case Replace is zero).

prints each line containing FOO. With an argument, it prints that many lines before and after each occurrence.

M-X Count Occurrences↓FOO<cr>
prints the number of occurrences of FOO after point.

M-X Delete Non-Matching Lines↓FOO<cr>
kills all lines after point that don't contain FOO.

M-X Delete Matching Lines↓FOO<cr>
kills all lines after point that contain FOO.

19.3. TECO Search Strings

The first string argument to Replace and Query Replace is actually a TECO search string. This means that the characters ↑X, ↑B, ↑N, ↑S, ↑O, and ↑Q have special meanings.

↑X matches any character. ↑B matches any "delimiter" character (anything which the word commands consider not part of a word, according to the syntax table. See section 22.4 [Syntax], page 119.). ↑N negates what follows, so that ↑N A matches anything but A, and ↑N ↑B matches any non-delimiter. ↑S is followed by a parameter character, and matches anything whose Lisp syntax equals the parameter. So ↑S(matches any character given the syntax of an open-parenthesis. ↑N↑SA matches any character which is not part of symbol names.

↑O means "or", so that XYXY ↑O ZZZ matches *either* XYXY or ZZZ. ↑O can be used more than once in a pattern. ↑Q quotes the following character, in case you want to search for one of the special control characters. However, you can't quote an Altmode or a Return in this way because its specialness is at an earlier stage of processing.

Some variables are supposed to have TECO search strings as their values. For example, Page Delimiter is supposed to be a search string to match anything which should start a page. This is so that you can use ↑O to match several alternatives. In the values of such variables, ↑B, ↑N, ↑O, ↑Q, ↑S, ↑X and ↑] are special, but Altmode is not. ↑B through ↑X are quoted with a ↑Q, and ↑] is quoted with another ↑].

The function Apropos (or, Help A) and all similar functions actually take TECO search strings as arguments, so you can search for more than one substring at a time. This is useful because doing Apropos on word↑Opara is not really slower than searching for just "word" or just "para".

Chapter Twenty

Editing Programs

Special features for editing programs include automatic indentation, comment alignment, parenthesis matching, and the ability to move over and kill balanced expressions. Many of these features are parameterized so that they can work for any programming language.

For each language there is a special *major mode* which customizes EMACS slightly to be better suited to editing programs written in that language. These modes sometimes offer special facilities as well.

See section 11.1 [Words], page 45. Moving over words is useful for editing programs as well as text.

See section 11.2 [Paragraphs], page 46. Most programming language major modes define paragraphs to be separated only by blank lines and page boundaries. This makes the paragraph commands useful for editing programs.

See section 21 [Tags], page 107. The TAGS package can remember all the labels or functions in a multi-file program and find any one of them quickly.

20.1. Major Modes

EMACS has many different *major modes*, each of which customizes EMACS for editing text of a particular sort. The major modes are mutually exclusive, and one major mode is current at any time. When at top level, EMACS always says in the mode line which major mode you are in.

When EMACS starts up, it is in what is called *Fundamental mode*, which means that the character commands are defined so as to be convenient in general. More precisely, in Fundamental mode every EMACS option is set in its default state. For editing any specific type of text, such as Lisp code or English text, you should switch to the appropriate major mode. This tells EMACS to change the meanings of a few commands to become more specifically adapted to the language being edited. Most commands remain unchanged; the ones which usually change are Tab, Rubout, and Linefeed. In addition, the commands which handle comments use the mode to determine how comments are to be delimited.

Selecting a new major mode is done with an M-X command. Each major mode is the name of the function to select that mode. Thus, you can enter Lisp mode by executing M-X Lisp (short for M-X Lisp Mode). You can specify which major mode

Command. This should be a TECO expression which takes complete responsibility for the compilation. It can find the filename to use in q-register 1. It *must* use ↑\ to exit. All the other hooks described above are ignored. This is often used when several input files must be compiled together in order to compile any of them. See the file AI:EMACS1;CCL > for an example of doing this for an EMACS library.

20.3. Indentation Commands for Code

Tab	Indents current line.
Linefeed	Equivalent to Return followed by Tab.
M-^	Joins two lines, leaving one space between if appropriate.
M-\	Deletes all spaces and tabs around point.
M-M	Moves to the first nonblank character on the line.

Most programming languages have some indentation convention. For Lisp code, lines are indented according to their nesting in parentheses. For assembler code, almost all lines start with a single tab, but some have one or more spaces as well. Indenting TECO code is an art rather than a science, but it is often useful to indent a line under the previous one.

Whatever the language, to indent a line, use the Tab command. Each major mode defines this command to perform the sort of indentation appropriate for the particular language. In Lisp mode, Tab aligns the line according to its depth in parentheses. No matter where in the line you are when you type Tab, it aligns the line as a whole. In MIDAS mode, Tab inserts a tab, that being the standard indentation for assembly code. In TECO mode, Tab realigns the current line to match a previous line. PL1 mode (See the file INFO;EPL1 >.) knows in great detail about the keywords of the language so as to indent lines according to the nesting structure.

The command Linefeed (^R Indent New Line) does a Return and then does a Tab on the next line. Thus, Linefeed at the end of the line makes a following blank line and supplies it with the usual amount of indentation. Linefeed in the middle of a line breaks the line and supplies the usual indentation in front of the new line.

The inverse of Linefeed is Meta-^ or C-M-^ (^R Delete Indentation). This command deletes the indentation at the front of the current line, and the line separator as well. They are replaced by a single space, or by no space if before a ")" or after a "(", or at the beginning of a line. To delete just the indentation of a line, go to the beginning of the line and use Meta-\ (^R Delete Horizontal Space), which deletes all spaces and tabs around the cursor.

To insert an indented line before the current one, do C-A, C-O, and then Tab. To make an indented line after the current one, use C-E Linefeed.

To move over the indentation on a line, use Meta-M or C-M-M (^R Back to Indentation). These commands move the cursor forward or back to the first nonblank character on the line.

The implementation of this feature uses the TECO flag FS ^R PAREN♦. See section 22.5 [FS Flags], page 121.

20.5. Manipulating Comments

The comment commands insert, kill and align comments. There are also commands for moving through existing code and inserting comments.

M-;	Insert or align comment.
C-;	The same.
C-M-;	Kill comment.
Return	Move past comment terminator and onto new line.
C-X ;	Set comment column.
M-N	Move to Next line and insert comment.
M-P	Move to Previous line and insert comment.
M-J	Continue a comment on a new line.
M-Linefeed	The same.

The command that creates a comment is Meta-; or Control-; (^R Indent for Comment). If there is no comment already on the line, a new comment is created, aligned at a specific column called the *comment column*. The comment is created by inserting whatever string EMACS thinks should start comments in the current major mode. Point is left after the comment-starting string. If the text of the line goes past the comment column, then the indentation is done to a suitable boundary (usually, a multiple of 8).

Meta-; can also be used to align an existing comment. If a line already contains the string that starts comments, then M-; just moves point after it and re-indents it to the right column. Exception: comments starting in column 0 are not moved.

Even when an existing comment is properly aligned, M-; is still useful for moving directly to the start of the comment.

Some languages require an explicit comment terminator, which is not simply the end of the line. Although the language may then allow comments in the middle of the line, the EMACS comment commands assume that a comment is the last thing on the line. When there is a comment terminator, M-; inserts the terminator as well as the starter, and leaves point between them, so that you are ready to insert the text of the comment. When you are done, the Return command given immediately before the comment terminator acts as if it were at the end of the line already: it moves down to or creates a following blank line. It does not break the existing line before the comment terminator as you would expect.

C-M-; (^R Kill Comment) kills the comment on the current line, if there is one. The indentation before the start of the comment is killed as well. If there does not appear to be a comment in the line, nothing is done. To reinsert the comment on another line, move to the end of that line, do C-Y, and then do M-; to realign it.

have any ";" recognized as starting a comment but have new comments begin with ";; ** ".

The string used to end a comment is kept in the variable Comment End. In many languages no comment end is needed as the comment extends to the end of the line. Then, this variable is a null string.

If Comment Multi Line is nonzero, then Meta-Linefeed within a comment does not close the old comment and start a new comment on the new line. Instead it allows the original comment to extend through the new line. This is legitimate if the language has explicit comment terminators. Then it's a matter of taste.

20.6. Lisp Mode and Muddle Mode

Lisp's simple syntax makes it much easier for an editor to understand; as a result, EMACS can do more for Lisp, and with less work, than for any other language.

Lisp programs should be edited in Lisp mode. In this mode, Tab is defined to indent the current line according to the conventions of Lisp programming style. It does not matter where in the line Tab is used; the effect on the line is the same. The function which does the work is called ^R Indent for Lisp. Linefeed, as usual, does a Return and a Tab, so it moves to the next line and indents it.

As in most modes where indentation is likely to vary from line to line, Rubout is redefined to treat a tab as if it were the equivalent number of space (^R Backward Delete Hacking Tabs). This makes it possible to rub out indentation one position at a time without worrying whether it is made up of spaces or tabs. Control-Rubout does the ordinary type of rubbing out which rubs out a whole tab at once.

Paragraphs are defined to start only with blank lines so that the paragraph commands can be useful. Auto Fill indents the new lines which it creates. Comments start with ";". If Atom Word mode is in effect, them in Lisp mode the word-motion commands regard each Lisp atom as one word.

The LEDIT library allows EMACS and Lisp to communicate, telling Lisp the new definitions of functions which you edit in EMACS. See the file INFO;LEDIT >.

The language Muddle is a variant form of Lisp which shares the concept of using parentheses (of various sorts) as the main syntactical construct. It can be edited using Muddle mode, which is almost the same as Lisp mode and provides the same features, differing only in the syntax table used.

20.6.1. Moving Over and Killing Lists and S-expressions

C-M-F	Move Forward over s-expression.
C-M-B	Move Backward.
C-M-K	Kill s-expression forward.
C-M-Rubout	Kill s-expression backward.
C-M-U	Move Up and backward in list structure.
C-M-(	The same.
C-M-)	Move up and forward in list structure.

To move *down* in list structure, use C-M-D (^R Down List). It is nearly the same as searching for a "(".

A somewhat random-sounding command which is nevertheless easy to use is C-M-T (^R Transpose Sexps), which drags the previous s-expression across the next one. An argument serves as a repeat count, and a negative argument drags backwards (thus canceling out the effect of C-M-T with a positive argument). An argument of zero, rather than doing nothing, transposes the s-expressions at the point and the mark.

To make the region be the next s-expression in the buffer, use or C-M-@ (^R Mark Sexp) which sets mark at the same place that C-M-F would move to. C-M-@ takes arguments like C-M-F. In particular, a negative argument is useful for putting the mark at the beginning of the previous s-expression.

The commands M-(("^R Insert ()") and M-) ("^R Move Over)") are designed for a style of editing which keeps parentheses balanced at all times. M-(inserts a pair of parentheses, either together as in "()", or, if given an argument, around the next several s-expressions, and leaves point after the open parenthesis. Instead of typing "(FOO)", you can type M-(FOO, which has the same effect except for leaving the cursor before the close parenthesis. Then you type M-), which moves past the close parenthesis, deleting any indentation preceding it (in this example there is none), and indenting with Linefeed after it.

The library LSPUTL contains two other list commands. Find Pat searches for lists which contain several strings. ^R Extract Sublist replaces a list with one of its sublists. See section 22.2 [Libraries], page 116.

<level>M-X Find Pat↓<mainstring>↓<string1>↓<string2> searches for a list which contains <mainstring> at a depth of <level> lists down, and contains <string1> and <string2> at any level. There can be any number of such additional strings to search for; there can be none of them.

^R Extract Sublist is meant to be connected to a character. Given an argument of <level>, it replaces the list <level> levels up from point with its sublist which starts after point.

The library XLISP contains a functions for making various transformations on Lisp code:

Lowercase Lisp Buffer changes all the Lisp code in the buffer to lower case, without changing comments, strings, or slashified characters. Uppercase Lisp Buffer performs the inverse transformation. Lowercase Lisp Region and Uppercase Lisp Region are similar but act only between point and mark.

Change / to \ takes Lisp code written with "/" as the character-quote character and transforms it to use the new character-quote character, "\". The meaning of the transformed code in the new syntax is the same as that of the old code in the old syntax.

XLISP contains several other commands which transform old constructs into new ones. They behave like Query Replace in that they display each occurrence of the old construct and ask you whether to change it to the new one. A Space means yes, a Rubout means no. Here is a list of these commands, and what each one transforms.

20.7. Lisp Grinding

The best way to keep Lisp code properly indented ("ground") is to use EMACS to re-indent it when it is changed. EMACS has commands to indent properly either a single line, a specified number of lines, or all of the lines inside a single s-expression.

Tab	In Lisp mode, re-indents line according to parenthesis depth.
Linefeed	Equivalent to Return followed by Tab.
M-^	Join two lines, leaving one space between them if appropriate.
C-M-Q	Re-indent all the lines within one list.
C-M-G	Grind a list, moving code between lines.

The basic indentation function is ^R Indent for Lisp, which gives the current line the correct indentation as determined from the previous lines' indentation and parenthesis structure. This function is normally found on C-M-Tab, but when in Lisp mode it is placed on Tab as well (Use Meta-Tab or C-Q Tab to insert a tab). If executed at the beginning of a line, it leaves point after the indentation; when given inside the text on the line, it leaves point fixed with respect to the characters around it.

When entering a large amount of new code, use Linefeed (^R Indent New Line), which is equivalent to a Return followed by a Tab. In Lisp mode, a Linefeed creates or moves down onto a blank line, and then gives it the appropriate indentation.

To join two lines together, use the Meta-^ or Control-Meta-^ command (^R Delete Indentation), which is approximately the opposite of Linefeed. It deletes any spaces and tabs at the front of the current line, and then deletes the line separator before the line. A single space is then inserted, if EMACS thinks that one is needed there. Spaces are not needed before a close parenthesis, or after an open parenthesis.

If you are dissatisfied about where Tab indents the second and later lines of an s-expression, you can override it. If you alter the indentation of one of the lines yourself, then Tab will indent successive lines of the same list to be underneath it. This is the right thing for functions which Tab indents unaesthetically.

When you wish to re-indent code which has been altered or moved to a different level in the list structure, you have several commands available. You can re-indent a specific number of lines by giving the ordinary indent command (Tab, in Lisp mode) an argument. This indents as many lines as you say and moves to the line following them. Thus, if you underestimate, you can repeat the process later.

You can re-indent the contents of a single s-expression by positioning point before the beginning of it and typing Control-Meta-Q (^R Indent Sexp). The line the s-expression starts on is not re-indented; thus, only the relative indentation with in the s-expression, and not its position, is changed. To correct the position as well, type a Tab before the C-M-Q.

Another way to specify the range to be re-indented is with point and mark. The command C-M-\ (^R Indent Region) applies Tab to every line whose first character is between point and mark. In Lisp mode, this does a Lisp indent.

A more powerful grind command which can move text between lines is C-M-G (^R Format Code). You might or might not like it. It knows in different ways about Lisp code and Macsyma code.

Whatever its value, the hook is called with two arguments, which are the position in the buffer of the most recent unclosed "(" and the buffer position of the beginning of the line to be indented. The buffer position of the cursor at the time the tab was typed is stored as an offset from Z in qZ, so Z-qZ is that buffer position. The hook should *not* modify the buffer. If it returns 0 or no value, the caller will use the horizontal cursor position of point as the column to indent to. Hence, to indent under the "O" in PROG, it is sufficient to jump to that character in the buffer and return. Alternatively, the hook can simply return the desired indentation column number as a value.

The hook should return a nonzero precomma value if following lines of the same expression should be indented individually. If it does not return a nonzero precomma value, the caller may remember the indentation of this line and indent following lines the same way.

If Lisp FOO Indent has a TECO expression as its value, the same interface conventions apply to it.

Different Lisp-like languages can select an entirely different set of indentation patterns by changing the value of the variable Lisp Indent Language. Normally the value of this variable is the string "Lisp". All the variables listed above with names beginning with "Lisp" actually should have names beginning with the Lisp Indent Language. Thus, if Lisp Indent Language is changed to "Muddle", then the indentation commands will look for variables Muddle Indent Offset, Muddle Indentation Hook, Muddle PROG Indent, etc.

20.8. Editing Assembly-Language Programs

MIDAS mode is designed for editing programs written in the MIDAS assembler. Major modes for other assemblers, such as PALX, also exist but differ only in the syntax table and in the name of the major mode hook that they will invoke. (There is also IBM370 mode, for 370 assembler, which is completely different. Refer to the self-documentation of the IBM370 library for information on it).

In MIDAS mode, comments start with ";", and "<" and ">" have the syntax of parentheses. In addition, there are five special commands which understand the syntax of instructions and labels. These commands are:

C-M-N	Go to Next label.
C-M-P	Go to Previous label.
C-M-A	Go to Accumulator field of instruction.
C-M-E	Go to Effective Address field.
C-M-D	Kill next word and its Delimiting character.

Two other commands with slightly different uses are

M-[	Move up to previous blank line.
M-]	Move down to next blank line.

Any line which is not indented and is not just a comment is taken to contain a label. The label is everything up to the first whitespace (or the end of the line). C-M-N (^R Go to Next Label) and C-M-P (^R Go to Previous Label) both position the cursor

PL1 mode is for editing PL/1 code, and causes Tab to indent an amount based on the previous statement type. The body of the implementation of PL1 mode is in the library PL1, which is loaded automatically when necessary. See the file INFO;EPL1 >.

PASCAL mode is similar to PL1 mode, for PASCAL. It is in the library called PASCAL. See the file INFO;EPASC >.

FORTRAN mode is implemented by the FORTRAN library. See the file INFO; EFORTRAN >.

There are major modes for many other languages, but documentation for them except that in the libraries themselves. Any volunteers to write some? Meanwhile, you can look at the documentation in the libraries. See section 22.2 [Libraries], page 116.

Chapter Twenty-One

The TAGS Package.

The TAGS package remembers the locations of the function definitions in a file and enables you to go directly to the definition of any function, without searching the whole file.

The functions of several files that make up one program can all be remembered together if you wish; then the TAGS package will automatically select the appropriate file as well.

21.1. How to Make a Tag Table for a Program

To use the TAGS package, you must create a tag table for the source file or files in your package. Normally, the tag table does not reside in any of those files, but in a separate tag table file which contains the names of the text files which it describes. Tag table files are generated by the :TAGS program. The same program can be used to update the tag table if it becomes very far out of date (slight inaccuracies do not matter). Tag tables for INFO files work differently; the INFO file contains its own tag table, which describes only that file. See section 21.8 [INFO tag tables], page 113, for how to deal with them.

The normal mode of operation of the :TAGS program is to read in an existing tag table and update it by rescanning the source files that it describes. The old tag table file itself tells :TAGS which source files to process. When making a new tag table you must start by making a skeleton. Then :TAGS is used to turn the skeleton into an accurate tag table.

A skeleton tag table is like a real one except that it is empty; there are no tags in it. It contains exactly this much data, for each source file that it is going to describe:

```
<filenames>
0,<language>
^_
```

The languages that :TAGS understands now are TECO, LISP, MIDAS, FAIL, PALX, MUDDLE, MACSYMA, TJ6, and R. MIDAS will do for MACRO-10 files. Any incompletely specified filenames will default to > and to the directory on which the tag table file itself is stored. The "0," *must* be present, since :TAGS expects that there will be a number in that place and will be completely confused if there is not. The CRLF after each ^_ also *must* be present. You can omit both the last ^_ and its CRLF together, however.

21.3. Jumping to a Tag

To jump to the definition of a function, use the command Meta-Period <tag name> <cr>. You will go straight to the definition of the tag. If the definition is in a different file then TAGS visits that file. If it is in the same file, TAGS leaves the mark behind and prints "^@" in the echo area.

You do not need to type the complete name of the function to be found; any substring will do. But this implies that sometimes you won't get the function you intended. When that happens, C-U Meta-Period will find the "next" function matching what you typed (next, in the order of listing in the tag table). Thus, if you want to find the definition of X-SET-TYPE-1 and you specify just TYPE-1, you might find X-READ-TYPE-1 instead. You could then type C-U Meta-Period until you reach X-SET-TYPE-1.

If you want to make sure you reach a precise function the first time, you should just include a character of context before and after its name. Thus, in a Lisp program, put a space before and after the function name. In a MIDAS program, put a linefeed before it and a colon after.

If Meta-Period is used before M-X Visit Tag Table has been done, it asks for the name of a tag table file. After you type this name and a <cr>, you type the name of the tag as usual. If the variable Tag Table Filenames exists, it specifies the defaults for the filename.

Typing an Altmode as the first character of the argument to Meta-Period allows you to switch to a new tag table. It prompts for the tag table filenames, then prompts again for the tag.

21.4. Other Operations on Tag Tables

21.4.1. Adding a New Function to a Tag Table

When you define a new function, its location doesn't go in the tag table automatically. That's because EMACS can't tell that you have defined a function unless you tell it by invoking the function ^R Add Tag. Since the operation of adding a tag to a tag table has proved not to be very necessary, this function no longer placed on any character, by default. You can invoke with M-X or connect it to a character if you like. For this section, let's assume you have placed it on C-X Period.

When you type the command C-X Period, the pointer should be on the line that introduces the function definition, after the function name and the punctuation that ends it. Thus, in a Lisp program, you might type "(DEFUN FOO " (note the space after FOO) and then type the C-X Period. In a MIDAS program, you might give the C-X Period after typing "FOO:". In a TECO program in EMACS format, you might type C-X Period after "!Set New Foo:!".

^R Add Tag modifies only the copy of the tag table loaded into EMACS. To modify the tag table file itself, you must cause it to be saved. M-X Save All Files is the easiest way to do this.

M-X Tags Query Replace does a Query Replace over all the files in a tag table. Like M-X Tags Search, it sets Control-. up to be a command to continue the Query Replace, in case you wish to exit, do some editing, and then resume scanning.

With Tags Find File set nonzero, Tags Search or Tags Query Replace could easily require more buffers than EMACS has room for. To prevent such a problem, they do not always put each file in a separate buffer. If Tags Search or Tags Query Replace wants to search a file which is already visited in some buffer, it uses the copy in that buffer. But if the file is not present, and Tags Find File is 1, then instead of visiting it in its own buffer, they visit it in a buffer named *Tags Search*. So at most one new buffer is created. If Tags Find File is 2, a new buffer is created for each file.

The library MQREPL enables you to use Next File to repeat a sequence of many Query Replace commands over a set of files, performing all the replacements on one file at a time.

21.4.4. Miscellaneous Applications of Tags

M-X List Tags♦<file><cr> lists all the tags in the specified file. Actually, all the files in the tag table whose names contain the string <file> are listed.

M-X Tags Apropos♦<pat><cr> lists all known tags whose names contain <pat>.

M-X Tags File List inserts in the buffer a list of the files known in the visited tag table.

M-X Tags Rescan runs :TAGS over the visited tag table and revisits it. This is the most convenient way to update the tag table.

M-X View Arglist♦<tag><cr> lets you look briefly at the line on which a tag is defined, and at the lines of comments which precede the definition. This is a good way to find out what arguments a function needs. The file is always loaded into a separate buffer, when this command is used.

M-X What Tag? tells you which function's definition you are in. It looks through the tag table for the tag which most nearly precedes point.

21.5. What Constitutes a Tag

In Lisp code, a function definition must start with an "(" at the beginning of a line, followed immediately with an atom which starts with "DEF" (and does not start with "DEFP"), or which starts with "MACRO", or which starts with "ENDF". The next atom on the line is the name of the tag. If there is no second atom on the line, there is no tag.

In MIDAS code, a tag is any symbol that occurs at the beginning of a line and is terminated with a colon or an equal sign. MIDAS mode is good for MACRO-10 also.

FAIL code is like MIDAS code, except that one or two ↓'s or "^"'s are allowed before a tag, spaces are allowed between the tag name and the colon or equal sign, and _ is recognized as equivalent to =.

new tag table. Then use :TAGS to update the tag table. The dummy will turn into a real entry.

You can delete a source file from a tag table by deleting its entire entry. Since the counts of the remaining entries are still valid, you need not run :TAGS over the file again. You can also change the order of the entries without doing any harm. The order of the entries matters if there are tags which appear in more than one source file.

You can edit everything else in the tag table too, if you want to. You might want to change a language name once in a while, but I doubt you will frequently want to add or remove tags, especially since that would all be undone by the next use of :TAGS!

21.7. How a Tag Is Described in the Tag Table

A tag table file consists of one or more entries in succession. Each entry lists the tags of one source file, and has the overall format described in the previous section, containing zero or more lines describing tags. Here we give the format of each of those lines.

Starting with the third line of the tag table entry, each line describes a tag. It starts with a copy of the beginning of the line that the tag is defined on, up through the tag name and its terminating punctuation. Then there is a rubout, followed by the character position in decimal of the place in the line where copying stopped. For example, if a line in a MIDAS program starts with "FOO:" and the colon is at position 602 in the file, then the line describing it in the tag table would be

```
FOO:<rubout>603
```

One line can describe several tags, if they are defined on the same line; in fact, in that case, they must be on the same line in the tag table, since it must contain everything before the tag name on its definition line. For example,

```
!Foo:! !Bar:!
```

in a file of TECO code followed by character number 500 of the file would turn into

```
!Foo:! !Bar:!<rubout>500
```

EMACS will be able to use that line to find either FOO or BAR. :TAGS knows how to create such things only for TECO files, at the moment. They aren't necessary in Lisp or MACSYMA files. In MIDAS files, :TAGS simply ignores all but the first tag on a line.

21.8. Tag Tables for INFO Structured Documentation Files

INFO files are divided up into nodes, which the INFO program must search for. Tag tables for these files are designed to make the INFO program run faster. Unlike a normal tag table, the tag table for an INFO file resides in that file and describes only that file. This is so that INFO, when visiting a file, can automatically use its tag table if it has one. INFO uses the tag tables of INFO files itself, without going through the normal TAGS package, which has no knowledge of INFO file tag tables. Thus, INFO

Chapter Twenty-Two

Simple Customization

In this chapter we describe the many simple ways of customizing EMACS without knowing how to write TECO programs.

One form of customization, reconnection of commands to functions, was discussed above in the explanation of how M-X commands work. See section 5.2.3 [Reconnecting Commands], page 23.

22.1. Minor Modes

Minor modes are options which you can use or not. For example, Auto Fill mode is a minor mode in which Spaces break lines between words as you type. All the minor modes are independent of each other and of the selected major mode. Most minor modes say in the mode line when they are on; for example, "Fill" in the mode line means that Auto Fill mode is on.

Each minor mode is the name of the function that can be used to turn it on or off. With no argument, the function turns the mode on if it was off and off if it was on. This is known as *toggling*. A positive argument always turns the mode on, and an explicit zero argument or a negative argument always turns it off. All the minor mode functions are suitable for connecting to single or double character commands if you want to enter and exit a minor mode frequently.

Auto Fill mode allows you to type text endlessly without worrying about the width of your screen. Line separators are be inserted where needed to prevent lines from becoming too long. See section 11.4 [Filling], page 50.

Auto Save mode protects you against system crashes by periodically saving the file you are visiting. Whenever you visit a file, auto saving is enabled if Auto Save Default is nonzero; in addition, M-X Auto Save allows you to turn auto saving on or off in a given buffer at any time. See section 13.3 [Auto Save], page 63.

Atom Word mode causes the word-moving commands, in Lisp mode, to move over Lisp atoms instead of words. Some people like this, and others don't. In any case, the s-expression motion commands can be used to move over atoms. If you like to use segmented atom names like FOOBAR-READ-IN-NEXT-INPUT-SOURCE-TO-READ, then you might prefer not to use Atom Word mode, so that you can use M-F to move over just part of the atom, or C-M-F to move over the whole atom.

Overwrite mode causes ordinary printing characters to replace existing text instead

or on request to make their functions available. See section [Catalogue], page 195, for a list of them.

To load a library, say M-X Load Library↓<libname><cr>. The library is found, either on your own directory or whichever one you specify, or on the EMACS directory, and loaded in. All the functions in the library are then available for use. Whenever you use M-X, the function name you specify is looked up in each of the libraries which you have loaded, more recently loaded libraries first. The first definition found is the one that is used.

For example, if you load the PICTURE library, you can then use M-X Edit Picture to run the Edit Picture function which exists in that library.

In addition to making functions accessible to M-X, the library may connect some of them to command characters. This is done by the library's & Setup function (See the file INFO;CONV >, node Lib.). If you give Load Library an argument, the setup is not done.

You can also load a library temporarily, just long enough to use one of the functions in it. This avoids taking up space permanently with the library. Do this with the function Run Library, as in M-X Run↓<libname>↓<function name><cr>. The library <libname> is loaded in, and <function name> executed. Then the library is removed from the EMACS job. You can load it in again later.

M-X List Loaded Libraries types the names and brief descriptions of all the libraries loaded, last loaded first. The last one listed is always the EMACS library.

You can get a brief description of all the functions in a library with M-X List Library↓<libname><cr>, whether the library is loaded or not. This is a good way to begin to find out what is in a library that has no INFO documentation. Continue by loading the library and using Help D to inquire further about whichever functions looked interesting.

The function Kill Libraries can be used to discard libraries loaded with Load Library. (Libraries used with Run Library are discarded automatically). However, of all the libraries presently loaded, only the most recently loaded one can be discarded. Kill Libraries offers to kill each loaded library, most recently loaded first. It keeps killing libraries until you say to keep one library. Then it returns, because the remaining libraries cannot be deleted if that library is kept.

Libraries are loaded automatically in the course of executing certain functions. You will not normally notice this. For example, the TAGS library is automatically loaded in whenever you use M-. or Visit Tag Table for the first time. This process is known as *autoloading*. It is used to make the functions in the TAGS library available without the user's having to know to load the library himself, while not taking up space in EMACSes of people who aren't using them. It works by simply calling Load Library on the library known to be needed. Another kind of autoloading loads a library temporarily, the way Run Library does. This is done when you use the DIRED function, for example, since the DIRED library is not needed after the DIRED function returns. (This does not use Run Library; it uses M.A, which is what Run Library uses).

You can make your own libraries, which you and other people can then use, if you know how to write TECO code. See the file INFO;CONV >, node Lib, for more details.

If you want to set a variable a particular way each time you use EMACS, you can use an init file or an EVARS file. This is one of the main ways of customizing EMACS for yourself. An init file is a file of TECO code to be executed when you start EMACS up. They are very general, but writing one is a black art. You might be able to get an expert to do it for you, or modify a copy of someone else's. See the file INFO;CONV >, node Init, for details. An EVARS file is a much simpler thing which you can do yourself. See section 22.6 [EVARS files], page 123.

You can also set a variable with the TECO command

```
<value> M.V <varname>$
```

or

```
:I*<string>$ M.V <varname>$
```

This is useful in init files.

Any variable can be made local to a specific buffer with the TECO command M.L<variable name>$. Thus, if you want the comment column to be column 50 in one buffer, whereas you usually like 40, then in the one buffer do M.LComment Column$ using the minibuffer. Then, you can do 50U$Comment Column$ in that buffer and other buffers will not be affected. This is how local modes lists in files work. M-X List Redefinitions describes the local variables of the selected buffer in a verbose fashion.

Most local variables are killed if you change major modes. Their global values come back. They are therefore called *mode locals*. There are also *permanent* locals which are not killed by changing modes; use 2,M.L to create one. Permanent locals are used by things like Auto Save mode to keep internal information about the buffer, whereas mode locals are used for customizations intended only for one buffer. See the file INFO;CONV >, node Variables, for information on how local variables work, and additional related features.

Local values of variables can be specified by the file being edited. For example, if a certain file ought to have a 50 column width, it can specify a value of 50 for the variable Fill Column. Then Fill Column will have the value 50 whenever this file is edited, *by anyone*. Editing other files is not affected. See section 22.7 [Locals], page 127, for how to do this.

22.4. The Syntax Table

All the EMACS commands which parse words or balance parentheses are controlled by the *syntax table*. Each ASCII character has a word syntax and a Lisp syntax. By changing the word syntax, you can control whether a character is considered a word delimiter or part of a word. By changing the Lisp syntax, you can control which characters are parentheses, which ones are parts of symbols, which ones are prefix operators, and which ones are just ignored when parsing s-expressions.

The syntax table is actually a string which is 128*5 characters long. Each group of 5 consecutive characters of the syntax table describe one ASCII character's syntax; but only the first three of each group are used. To edit the syntax table, use M-X Edit

groups. You can see easily what the syntax of any character is. You are not editing the table immediately, however. Instead, you are asked for the character whose syntax you wish to edit. After typing it, you are positioned at that character's five-character group. Overwrite mode is on, so you can simply type the desired syntax entries, which replace the old ones. You can also do arbitrary editing, but be careful not to change the position of anything in the buffer. When you exit the recursive editing level, you are asked for another character to position to. An Altmode at this point exits and makes the changes. A C-] at any time aborts the operation.

Many major modes alter the syntax table. Each such major mode creates its own syntax table once and reselects the same string whenever the mode is selected, in any buffer. Thus, all buffers in Text mode at any time use the same syntax table. This is important because if you ever change the syntax table of one buffer that is in Text mode, you change them all. It is possible to give one buffer a local copy with a TECO program:

```
MM Make Local Q-Register♦..D♦W  :G..DU..D
```

The syntax tables belonging to the major modes are not preinitialized in EMACS; they are created when the major mode is invoked for the first time, by copying the default one and making specific changes. Thus, any other changes you have made in the default (Fundamental mode) syntax table at the beginning propagate into all modes' syntax tables unless those modes specifically override them.

After a major mode has created its own syntax table, that table is stored in the variable <modename> ..D. This makes a different variable for each major mode, since the mode name is part of the variable name. Further use of the major mode gets the syntax table from that variable. If you create the variable yourself before the first use of the major mode, the value you put there will be used.

TECO programs and init files can most easily change the syntax table with the function & Alter ..D (look at its documentation). The syntax table is kept in the q-register named ..D, which explains that name.

22.5. FS Flags

FS flags are variables defined and implemented by TECO below the level of EMACS. Some of them are options which control the behavior of parts of TECO such as the display processor. Some of them control the execution of TECO programs; you are not likely to want to change these. Others simply report information from inside TECO. The list of FS flags is fixed when TECO is assembled and each one exists for a specific purpose.

FS flags are used mostly by the TECO programmer, but some of them are of interest to the EMACS user doing minor customization. For example, FS ECHO LINES♦ is the number of lines in the echo area. By setting this flag you can make the echo area bigger or smaller. Many FS flags useful for customization are controlled by EMACS variables; instead of setting the FS flag, you can set the EMACS variable like any other. Setting the variable automatically sets the FS flag as well. Here is a list of such variables which control flags:

22.6. Init Files and EVARS Files

EMACS is designed to be customizable; each user can rearrange things to suit his taste. Simple customizations are primarily of two types: moving functions from one character to another, and setting variables which functions refer to so as to direct their actions. Beyond this, extensions can involve redefining existing functions, or writing entirely new functions and creating sharable libraries of them.

The most general way to customize is to write an init file, a TECO program which is executed whenever you start EMACS. The init file is found by looking for a particular filename, <home directory>;<user name> EMACS. This method is general because the program can do anything. It can ask you questions and do things, rather than just setting up commands for later. However, TECO code is arcane, and only a few people learn how to write it. If you need an init file and don't feel up to learning to write TECO code, ask a local expert to do it for you. See the file INFO;CONV >, for more about init files.

However, simple customizations can be done in a simple way with an EVARS file. Such a file serves the same sort of purpose as an init file, but instead of TECO code, it contains just a list of variables and values. Each line of the EVARS file names one variable or one command character and says how to redefine it. Empty lines, and lines starting with spaces or tabs, are ignored. They can be used as comments. Your EVARS file is found by its filename, as an init file is, but it should be called <home directory>;<user name> EVARS instead of EMACS. You can have both an init file and an EVARS file if you want, as long as your init file calls the default init file, since that is what processes the EVARS file.

To set a variable, include in the EVARS file a line containing the name of the variable, a colon, and the value. A numeric value is represented by the number. A string value is enclosed in double quotes. To include a double quote or a ↑] character in the value of the string, precede it with a ↑] to quote it. You can also simply give the string value, with no quotes, as long as it is not ambiguous (does not consist of digits or start with a double quote); however, in this case, any spaces following the colon become part of the value of the variable. They are not ignored. Examples:

```
Comment Column: 70
Comment Start: ";"
MM Foo:FTFOO$
```

The last line defines a variable named MM Foo, which has the effect of defining a function named Foo with the specified value as its definition.

To redefine a command character is a little more complicated. Instead of the name of a variable, give a ↑R (control-R) followed by the character. Since the general Control and Meta character cannot be part of a file, all Control and Meta characters are represented in a funny way: after the ↑R put the residue of the character after removing the Control and Meta, and before the ↑R put periods, one for Control, two for Meta, and three for Control-Meta. Thus, C-D is represented by ".↑RD" and C-M-; is represented by "...↑R;". Lower case characters such as C-a are usually defined as "execute the definition of the upper case equivalent". Therefore, by redefining the C-A command you also change C-a; but if you redefine C-a, by saying ".↑Ra" instead of ".↑RA", you will not change C-A. So be careful about case.

22.6.1. EVARS File Examples

Here are some examples of how to do various useful things in an EVARS file.

This causes new buffers to be created in Lisp mode:

```
Default Major Mode: "LISP"
```

This causes new buffers to have Auto Fill mode turned on:

```
Buffer Creation Hook: "1M.L Auto Fill Mode◆"
```

This causes all Text mode buffers to have Auto Fill mode turned on:

```
Text Mode Hook: "1M.L Auto Fill Mode◆"
```

This causes C-M-G to be undefined by copying the definition of C-| (which is undefined):

```
...↑RG: Q.↑R|
```

This redefines C-S to be a single character search command, and M-S to be a non-incremental string search:

```
.↑RS: M.M ^R Character Search◆
..↑RS: M.M ^R String Search◆
```

This redefines C-X V to run View File:

```
:.X(↑^V): M.M View File◆
```

This makes M-M a prefix character and defines M-M W to mark a word and M-M P to mark a paragraph. It stores the dispatch vector for the prefix character in the variable M-M Dispatch.

```
..↑RM: MM Make Prefix Character◆≥◆M-M Dispatch≥◆◆
Temp: "M-M  ◆M-M Dispatch◆
"
   Append the line in Temp to Prefix Char List.
*: Q◆Prefix Char List◆[1 Q◆Temp◆[2 :i◆Prefix Char List◆↑]1↑]2◆
:◆M-M Dispatch◆(↑^W): M.M ^R Mark Word◆
:◆M-M Dispatch◆(↑^P): M.M ^R Mark Paragraph◆
```

This loads the library LUNAR and defines C-Q to run a useful function found in that library:

```
*: MM Load Library◆LUNAR◆
.↑RQ: M.M ^R Various Quantities◆
```

This causes Auto Save mode to save under the visited filenames:

```
Auto Save Visited File: 1
```

This causes TAGS to bring new files into separate buffers:

```
TAGS Find File: 1
```

This prevents the default init file from printing the message "EMACS version nnn. Type ... for Help".

```
Inhibit Help Message: 1
```

This redefines the list syntax of "%" to be ";" for "comment starter", and that of ";" to be "A" for "alphabetic":

This causes TAGS to bring new files into separate buffers:

```
1M.VTAGS Find File♦
```

This prevents the default init file from printing the message "EMACS version nnn. Type ... for Help".

```
1M.VInhibit Help Message♦
```

This redefines the list syntax of "%" to be ";" for "comment starter", and that of ";" to be "A" for "alphabetic":

```
1mm& Alter ..D♦%;;A♦
```

22.7. Local Variables in Files

By specifying *local modes* in a file you can cause certain major or minor modes to be set, or certain character commands to be defined, whenever you are visiting it. For example, EMACS can select Lisp mode for that file, set up a special Comment Column, or put a special command on the character C-M-Comma. Local modes can specify the major mode, and the values of any set of named variables and command characters. Local modes apply only while the buffer containing the file is selected; they do not extend to other files loaded into other buffers.

The simplest kind of local mode specification sets only the major mode. You put the mode's name in between a pair of "-*-"'s, anywhere on the first nonblank line of the file. For example, the first line of this file contains -*-Scribe-*-, implying that this file should be edited in Scribe mode.

To specify more that just the major mode, you must use a *local modes list*, which goes in the *last* page of the file (it is best to put it on a separate page). The local modes list starts with a line containing the string "Local Modes:", and ends with a line containing the string "End:". In between come the variable names and values, just as in an EVARS file. See section 22.6 [EVARS files], page 123.

The line which starts the local modes list does not have to say just "Local Modes:". If there is other text before "Local Modes:", that text is called the *prefix*, and if there is other text after, that is called the *suffix*. If these are present, each entry in the local modes list should have the prefix before it and the suffix after it. This includes the "End:" line. The prefix and suffix are included to disguise the local modes list as a comment so that the compiler or text formatter will not be perplexed by it. If you do not need to disguise the local modes list as a comment in this way, do not bother with a prefix or a suffix.

Aside from the "Local Modes:" and the "End:", and the prefix and suffix if any, a local modes list looks like an EVARS file. However, comments lines are not allowed, and you cannot redefine C-X subcommands due to fundamental limitations of the data structure used to remember local variables. Sorry.

The major mode can be set by specifying a value for the variable "Mode" (don't try setting the major mode this way except in a local modes list!). It should be the first thing in the local modes list, if it appears at all. A function M-X Foo can be defined locally by putting in a local setting for a variable named "MM Foo". See section 5.2 [Functions], page 21.

editor. We decided to change the terminology because, when thinking of EMACS, we consider TECO a programming language rather than an editor. The only "macros" in EMACS now are keyboard macros.

You define a keyboard macro while executing the commands which are the definition. Put differently, as you are defining a keyboard macro, the definition is being executed for the first time. This way, you can see what the effects of your commands are, so that you don't have to figure them out in your head. When you are finished, the keyboard macro is defined and also has been, in effect, executed once. You can then do the whole thing over again by invoking the macro.

22.8.1. Basic Use

To start defining a keyboard macro, type the C-X (command (^R Start Kbd Macro). From then on, your commands continue to be executed, but also become part of the definition of the macro. "Def" appears in the mode line to remind you of what is going on. When you are finished, the C-X) command (^R End Kbd Macro) terminates the definition (without becoming part of it!).

The macro thus defined can be invoked again with the C-X E command (^R Execute Kbd Macro), which may be given a repeat count as a numeric argument to execute the macro many times. C-X) can also be given a repeat count as an argument, in which case it repeats the macro that many times right after defining it, but defining the macro counts as the first repetition (since it is executed as you define it). So, giving C-X) an argument of 2 executes the macro immediately one additional time. An argument of zero to C-X E or C-X) means repeat the macro indefinitely (until it gets an error).

If you want to perform an operation on each line, then either you should start by positioning point on the line above the first one to be processed and then begin the macro definition with a C-N, or you should start on the proper line and end with a C-N. Either way, repeating the macro will operate on successive lines.

After you have terminated the definition of a keyboard macro, you can add to the end of its definition by typing C-U C-X (. This is equivalent to plain C-X (followed by retyping the whole definition so far. As a consequence it re-executes the macro as previously defined.

If you wish to save a keyboard macro for longer than until you define the next one, you must give it a name. If you do M-X Name Kbd Macro◆FOO<cr>, the last keyboard macro defined (the one which C-X E would invoke) is turned into a function and given the name FOO. M-X FOO will from then on invoke that particular macro. Name Kbd Macro also reads a character from the keyboard and redefines that character command to invoke the macro. You can use a bit prefix character in specifying the command; you can also type a C-X command to be redefined. When you have finished typing the command characters, Name Kbd Macro asks you whether it should go ahead and redefine the character.

To save a keyboard macro permanently, do M-X Write Kbd Macro. Supply the function name of the keyboard macro as a string argument, or else it will ask you to type the character which invokes the keyboard macro. The keyboard macro is saved as a library which, when loaded, automatically redefines the keyboard macro. The

Chapter Twenty-Three

The Minibuffer

The *minibuffer* is a facility by means of which EMACS commands can read input from the terminal, allowing you to use EMACS commands to edit the input while you are typing it. Usually it is used to read a TECO program to be executed.

M-Altmode	Invokes an empty minibuffer.
M-%	Invokes a minibuffer initialized with a Query Replace.
C-X Altmode	Re-execute a recent minibuffer command.
C-X ^	Add more lines to the minibuffer.
C-\	Meta-prefix for use in the minibuffer.
C-C C-Y	Rotate ring of recent minibuffer commands.

The primary use of the minibuffer is for editing and executing simple TECO programs such as

```
MM Query Replace◆FOO
◆BAR
◆
```

(which could not be done with M-X Query Replace because when M-X is used Return terminates the arguments).

You can always tell when you are in a minibuffer, because the mode line contains something in parentheses, such as "(Minibuffer)" or "(Query Replace)". There is also a line of dashes across the screen a few lines from the top. Strictly speaking, the minibuffer is actually the region of screen above the line of dashes, for that is where you edit the input that the minibuffer is asking you for. Editing has been limited to a few lines so that most of the screen can continue to show the file you are visiting.

If you want to type in a TECO command, use the minibuffer with the command Meta-Altmode, (^R Execute Minibuffer). An empty minibuffer will appear, into which you should type the TECO command string. Exit with Altmode Altmode, and remember that neither of the two Altmodes is inserted into your TECO command although the first one may appear to be. When the TECO command is executed, "the buffer" will be the text you were editing before you invoked the minibuffer.

Often, a minibuffer starts out with some text in it. This means that you are supposed to add to that text, or, sometimes, to delete some of it so as to choose among several alternatives. For example, Meta-% (^R Query Replace) provides you with a minibuffer initially containing the string "MM Query Replace◆". The cursor comes at the end. You are then supposed to add in the arguments to the Query Replace.

In a minibuffer, you can edit your input until you are satisfied with it. Then you tell

Chapter Twenty-Four

Correcting Mistakes and EMACS Problems

If you type an EMACS command you did not intend, the results are often mysterious. This chapter tells what you can do to cancel your mistake or recover from a mysterious situation. EMACS bugs and system crashes are also considered.

24.1. Quitting and Aborting

C-G Quit. Cancel running or partially typed command.

C-] Abort recursive editing level and cancel the command which invoked it.

M-X Top Level
Abort all recursive editing levels and subsystems which are currently executing.

There are three ways of cancelling commands which are not finished executing: *quitting* with C-G, and *aborting* with C-] or M-X Top Level. Quitting is cancelling a partially typed command or one which is already running. Aborting is cancelling a command which has entered a recursive editing level or subsystem.

Quitting with C-G is used for getting rid of a partially typed command, or a numeric argument that you don't want. It also stops a running command in the middle in a relatively safe way, so you can use it if you accidentally give a command which takes a long time. In particular, it is safe to quit out of killing; either your text will *all* still be there, or it will *all* be in the kill ring (or maybe both). Quitting an incremental search does special things documented under searching; in general, it may take two successive C-G's to get out of a search. C-G can interrupt EMACS at any time, so it is not an ordinary command.

Aborting with C-] (Abort Recursive Edit) is used to get out of a recursive editing level and cancel the command which invoked it. Quitting with C-G cannot be used for this, because it is used to cancel a partially typed command *within* the recursive editing level. Both operations are useful. For example, if you are editing a message to be sent, C-G can be used to cancel the commands you use to edit the message, and C-] cancels sending the message. C-] either tells you how to resume the aborted command or queries for confirmation before aborting.

When you are in a position to use M-X, you can use M-X Top Level. This is

24.2.3. Garbage on the Screen

If the data on the screen looks wrong, it could be due to line noise on input or output, a bug in the terminal, a bug in EMACS redisplay, or a bug in an EMACS command. To find out whether there is really anything wrong with your text, the first thing to do is type C-L. This is a command to clear the screen and redisplay it. Often this will display the text you expected. Think of it as getting an opinion from another doctor.

24.2.4. Garbage Displayed Persistently

If EMACS persistently displays garbage on the screen, or if it outputs the right things but scattered around all the wrong places on the screen, it may be that EMACS has the wrong idea of your terminal type. The first thing to do in this case is to exit from EMACS and restart it. Each time EMACS is restarted it asks the system what terminal type you are using. Whenever you detach and move to a terminal of a different type you should restart EMACS as a matter of course. If you stopped EMACS with the exit command, or by interrupting it when it was awaiting a command, then this is sure to be safe.

The system itself may not know what type of terminal you have. You should try telling the system with the :TCTYP command.

24.2.5. URK Error (Address Space Exhausted)

If attempting to visit a file or load a library causes an "URK" error, it means you have filled up the address space; there is no room inside EMACS for any more files or libraries. In this situation EMACS will try to run the function Make Space for you. If EMACS is unable to do it for you, you may still be able to do M-X Make Space yourself. This command compacts the data inside EMACS to free up some space. It also offers to discard data that may be occupying a lot of space, such as the kill ring (See section 9.1 [Killing], page 35.), the undo memory (See section 24.3 [Undo], page 136.), and buffers created by RMAIL, TAGS and INFO. Another way of freeing space is to kill buffers with M-X Kill Some Buffers (See section 14 [Buffers], page 71.) or unload libraries with M-X Kill Libraries (See section 22.2 [Libraries], page 116.).

Use the command M-X What Available Space to find out how close you are to running out of space. It tells you how many K of space you have available for additional files or libraries.

Visiting a file causes an URK error if the file does not fit in the available virtual memory space, together with the other buffers and the libraries loaded. A big enough file causes an URK error all by itself. For editing such large files, use the command Split File (in the SPLIT library) to break it into subfiles. These will be fairly large files still, but not too large to edit. After editing one or more of the subfiles, use the command Unsplit File (also in SPLIT) to put them back together again.

another, but it doesn't, because you can *always* Undo the Undo if it didn't help. But you can avoid even having to do that, if you look at what type of change Undo says it will undo.

If you want to undo a considerable amount of editing, not just the last change, the Undo command can't help you, but M-X Revert File (See section 13.2 [Revert], page 63.) might be able to. If you have been writing a journal file (See section 24.4 [Journals], page 137.), you can replay the journal after deleting the part that you don't want.

24.4. Journal Files

A journal file is a record of all the commands you type during an editing session. If you lose editing because of a system crash, an EMACS bug, or a mistake on your part, and you have made a journal file, you can replay the journal or part of it to recover what you lost. Journal files offer an alternative to auto saving, using less time and disk space if there is no crash, but requiring more time when you recover from a crash. See section 13.3 [Auto Save], page 63.

24.4.1. Writing Journal Files

In order to make a journal file, you must load the JOURNAL library and then execute M-X Start Journal File♦<filename><cr>. Immediately, most of the current status of EMACS is recorded in the journal file, and all subsequent commands are recorded as they are typed. This happens invisibly and silently. The journal file is made fully up to date on the disk after every 50th character, so the last 50 characters of type in is the most you can lose.

The default filenames for the journal file are <home directory>;<user name> JRNL. There is rarely a reason to use any other name, because you only need one journal file unless you are running two EMACSes at the same time.

24.4.2. Replaying Journal Files

To replay the journal file, get a fresh EMACS, load JOURNAL, and do M-X Replay Journal File♦<filename><cr>. The filename can usually be omitted since normally you will have used the defaults when creating the journal.

After a delay while the files, buffers and libraries are loaded as they were when the journal file was written, EMACS will begin replaying the commands in the journal before your very eyes. Unlike keyboard macros, which execute invisibly until they are finished, journal files display as they are executed. This allows you to see how far the replay has gone. You can stop the process at any time by typing C-G. Aside from that, you should not type anything on the keyboard while the replay is going on.

If the need for a replay is the result of a system crash or EMACS crash, then you probably want to replay the whole file. This is what happens naturally. If you are replaying because you made a great mistake, you probably want to stop the replay

Colons are also used to record the precise effects of certain commands such as C-V whose actions depend on how the text was displayed on the screen. Since the effects of such commands are not completely determined by the text, replaying the command could produce different results, especially if done on a terminal with a different screen size. The extra information recorded in the journal makes it possible to replay these commands with fidelity.

A semicolon in the journal file begins a comment, placed there for the benefit of a human looking at the journal. The comment ends at the beginning of the following line.

24.4.4. Warnings

Proper replaying of a journal file requires that all the surrounding circumstances be unchanged.

In particular, replaying begins by visiting all the files that were visited when the writing of the journal file began; not the latest versions of these files, but the versions which were the latest at the earlier time. If those versions, which may no longer be the latest, have been deleted, then replaying is impossible.

If your init file has been changed, the commands when replayed may not do what they did before.

These are the only things that can interfere with replaying, as long as you start writing the journal file immediately after starting EMACS. But as an editing session becomes longer and files are saved, the journal file contains increasing amounts of waste in the form of commands whose effects are already safe in the newer versions of the edited files. Replaying the journal will replay all these commands wastefully to generate files identical to those already saved, before coming to the last part of the session which provides the reason for replaying. Therefore it becomes very desirable to start a new journal file. However, many more precautions must be taken to insure proper replaying of a journal file which is started after EMACS has been used for a while. These precautions are described here. If you cannot follow them, you must make a journal checkpoint (see below).

If any buffer contains text which is not saved in a file at the time the journal file is started, it is impossible to replay the journal correctly. This problem cannot possibly be overcome. To avoid it, M-X Start Journal File offers to save all buffers before actually starting the journal.

Another problem comes from the kill ring and the other ways in which EMACS remembers information from previous commands. If any such information which originated before starting the journal file is used after starting it, the journal file cannot be replayed. For example, suppose you fill a paragraph, start a journal file, and then do M-X Undo? When the journal is replayed, it will start by doing M-X Undo, but it won't know what to undo. It is up to you not to do anything that would cause such a problem. It should not be hard. It would be possible to eliminate this problem by clearing out all such data structures when a journal file is started, if users would prefer that.

A more difficult problem comes from customization. If you change an option or

what it ought to have done. If you aren't familiar with the command, or don't know for certain how the command is supposed to work, then it might actually be working right. Rather than jumping to conclusions, show the problem to someone who knows for certain.

Finally, a command's intended definition may not be best for editing with. This is a very important sort of problem, but it is also a matter of judgement. Also, it is easy to come to such a conclusion out of ignorance of some of the existing features. It is probably best not to complain about such a problem until you have checked the documentation in the usual ways (INFO and Help), feel confident that you understand it, and know for certain that what you want is not available. If you feel confused about the documentation instead, then you don't have grounds for an opinion about whether the command's definition is optimal. Make sure you read it through and check the index or the menus for all references to subjects you don't fully understand. If you have done this diligently and are still confused, or if you finally understand but think you could have said it better, then you have a constructive complaint to make *about the documentation*. It is just as important to report documentation bugs as program bugs.

24.5.2. How to Report a Bug

When you decide that there is a bug, it is important to report it and to report it in a way which is useful. What is most useful is an exact description of what commands you type, starting with a fresh EMACS just loaded, until the problem happens. Send the bug report to BUG-EMACS@MIT-AI.

The most important principle in reporting a bug is to report *facts*, not hypotheses or conditions. It is always easier to report the facts, but people seem to prefer to strain to think up explanations and report them instead. If the explanations are based on guesses about how EMACS is implemented, they will be useless; we will have to try to figure out what the facts must have been to lead to such speculations. Sometimes this is impossible. But in any case, it is unnecessary work for us.

For example, suppose that you type C-X C-V GLORP;BAZ UGH<cr>, visiting a file which (you know) happens to be rather large, and EMACS prints out "I feel pretty today". The best way to report the bug is with a sentence like the preceding one, because it gives all the facts and nothing but the facts.

Do not assume that the problem is due to the size of the file and say "When I visit a large file, EMACS prints out 'I feel pretty today'". This is what we mean by "guessing explanations". The problem is just as likely to be due to the fact that there is a "Z" in the filename. If this is so, then when we got your report, we would try out the problem with some "big file", probably with no "Z" in its name, and not find anything wrong. There is no way in the world that we could guess that we should try visiting a file with a "Z" in its name.

Alternatively, the problem might be due to the fact that the file starts with exactly 25 spaces. For this reason, you should make sure that you don't change the file until we have looked at it. Suppose the problem only occurs when you have typed the C-X C-A command previously? This is why we ask you to give the exact sequence of characters you typed since loading the EMACS.

Chapter Twenty-Five

Word Abbreviation Input

Word Abbrev mode allows you to abbreviate text with a single "word", with EMACS expanding the abbreviation automatically as soon as you have finished the abbreviation.

Abbrevs are also useful for correcting commonly misspelled or mistyped words ("thier" could expand to "their"), and for uppercasing words like "EMACS" (abbrev "emacs" could expand to "EMACS").

To use this mode, just do M-X Word Abbrev Mode<cr>. (Another M-X Word Abbrev Mode<cr> will turn the mode off; it toggles.)

For example, in writing this documentation we could have defined "wam" to be an abbreviation for "word abbrev mode". After typing just the letters "wam", we see just that. "wam", but if we then finish the word by typing space or period or any other punctuation, the "wam" is replaced by (and redisplays as) "word abbrev mode". If we capitalize the abbrev, "Wam", the expansion is capitalized: "Word abbrev mode". If we capitalize the whole abbrev, WAM", each word in the expansion is capitalized: "Word Abbrev Mode". In this particular example, though, we would define "wam" to expand to "Word Abbrev mode" since it is always to be capitalized that way.

Thus, typing "I am in wam now" produces "I am in Word Abbrev mode now".

Word Abbrev mode does not interfere with the use of major modes, such as Text, Lisp, TECO, PL1, or minor modes, such as Auto Fill. Those modes (or the user) may redefine what functions are connected to characters; this does not hamper Word Abbrev mode.

There are two kinds of word abbreviations: mode and global. A mode word abbrev applies only in one major mode (for instance only in Text mode), while a global word abbrev applies regardless of major mode. If some abbrev is defined both as a mode word abbrev for the current mode and as a global word abbrev, the mode word abbrev expansion takes precedence.

For instance, you might want an abbrev "foo" for "find outer otter" in Text mode, an abbrev "foo" for "FINAGLE-OPPOSING-OPINIONS" in Lisp, and an abbrev "foo" for "meta-syntactic variable" in any other mode (the global word abbrev).

Word abbrevs can be defined one at a time (adding them as you think of them), or many at a time (from a definition list). You can save them in a file and read them back later. If you turn off Word Abbrev mode, abbrevs stop expanding automatically, but their definitions are remembered in case you turn Word Abbrev mode back on.

Sometimes you may think you already had an abbrev for some text, use it, and see that it didn't expand. In this case, the C-X C-H (^R Inverse Add Mode Word Abbrev) or C-X - (^R Inverse Add Global Word Abbrev) commands are helpful: they ask you to type in an *expansion* rather than an abbrev. In addition to defining the abbrev, they also expand it. If you give them a numeric argument, n, they use the nth word before point as the abbrev.

You can kill abbrevs (cause them to no longer expand) by giving a negative numeric argument to C-X C-A or C-X +. For instance, to kill the global abbrev "foo" type C-U - C-X + foo<cr>.

25.1.2. Controlling Abbrev Expansion

When an abbrev expands, the capitalization of the expansion is determined by the capitalization of the abbrev: If the abbrev is all lowercase, the expansion is as defined. If the abbrev's first letter is uppercase, the expansion's first letter is too. If the abbrev is all uppercase, there are two possibilities: if the expansion is a single word, it is all-uppercased; otherwise, each of its words has its first letter uppercased (such as for use in a title). (If you don't like this distinction between single-word and multi-word expansions, set the variable WORDAB All Caps to 1. Then an all-uppercase abbrev will always result in an all-uppercase expansion.)

Abbrevs normally expand when you type some punctuation character; the abbrev expands and the punctuation character is inserted. There are other ways of expanding abbrevs: C-M-Space (^R Abbrev Expand Only) causes the abbrev just before point to be expanded without inserting any other character. C-M-Space will expand abbrevs even if Word Abbrev mode is currently off; this can be useful if the system is slow, and you just want to manually expand a few abbrevs. M-' (^R Word Abbrev Prefix Mark) allows you to "glue" an abbrev onto any prefix: suppose you have the abbrev "comm" for "committee", and wish to insert "intercommittee "; type "inter", M-' (you will now see "inter-"), and then "comm "; "inter-comm " becomes "intercommittee ". M-X Expand Word Abbrevs in Region checks each word in the region and offers to expand each word abbrev found; for more details see its self-documentation. (It is similar to the M-X Query Replace command.)

25.1.3. Unexpanding Abbrevs

C-X U (^R Unexpand Last Word) "unexpands" the last abbrev's expansion, replacing the last expansion with the abbrev that caused it. If any auto-filling was done because of the expansion (you had Auto Fill mode on), that too is undone. If you type another C-X U, the first one is "undone" and the abbrev is expanded again. Only the last expansion can be undone. Sometimes you may find that C-X U unexpands an abbrev later than the one you're looking at. In this case, do another C-X U and go back and manually correct the earlier expansion.

If you know beforehand that a word will expand, and want to prevent it, you can simply "quote" the punctuation character with C-Q. For example, typing "comm", a C-Q, and then "." gives "comm." without expanding.

```
*: 1 MM Word Abbrev Mode♦
*: MM Read Word Abbrev File♦WORDAB DEFNS♦

Save Word Abbrevs:1
```

Or if you have an init file, use the following Teco code:

```
1 MM Word Abbrev Mode♦
MM Read Word Abbrev File♦WORDAB DEFNS♦

1u♦Save Word Abbrevs♦
```

25.2. Advanced Usage

The use of Word Abbrev mode as discussed in the previous section suffices for most users. However, some users who use Word Abbrev mode a lot or have highly tailored environments may desire more flexibility or need more power to handle extreme situations than the basic commands provide.

25.2.1. Alternatives and Customizations

M-X Make Word Abbrev♦<abbrev>♦<expansion>♦<mode><cr>

M-X Kill All Word Abbrevs<cr>

M-X Make These Characters Expand♦<characters><cr>

M-X Attach Word Abbrev Keyboard Macro

^R Kill Mode Word Abbrev

^R Kill Global Word Abbrev

Only Global Abbrevs
: Set this option if you only use globals.

Additional Abbrev Expanders
: Variable for adding a few more expanders.

WORDAB Ins Chars
: Variable for replacing entire set of expanders.

The basic commands for defining a new mode abbrev, C-X C-A (^R Add Mode Word Abbrev) and C-X C-H (^R Inverse Add Mode Word Abbrev), work only in the current mode. A more general command is M-X Make Word Abbrev which takes three string arguments: the first is the abbrev, the second is the expansion, and the third is the mode (such as "Text"). This command can also define global abbrevs, by providing "*" as the mode name.

M-X Kill All Word Abbrevs<cr> is a very quick way of killing every abbrev currently defined. After this command, no abbrev will expand. (A slower but more careful way is with M-X Edit Word Abbrevs.)

The functions ^R Kill Mode Word Abbrev and ^R Kill Global Word Abbrev exist, but are not connected to any commands by default. If you find having to specify negative arguments to C-X C-A (^R Add Mode Word Abbrev) and C-X + (^R Add Global Word

argument is actually a TECO search string (See section 19.3 [TECO search strings], page 89.). If you want to see the abbrevs which contain either <string1> *or* <string2>, separate the strings with a ↑O; to see abbrev definitions containing either "defn" or "wab", do M-X List Word Abbrevs♦defn↑Owab<cr>.

You can provide M-X List Word Abbrevs with an argument to control whether the filtering string applies to just the abbrev (C-U 1), just the expansion (C-U 2), just the mode (C-U 4), or any combination (the sum). C-U 3 M-X List Word Abbrevs♦lisp<cr> will match "lisp" against abbrevs and expansions, but not modes.

M-X Insert Word Abbrevs♦<string><cr> works similarly, but inserts the list into the buffer instead of typing it out.

25.2.4. Dumped EMACS Environments

M-X Write Word Abbrev File♦<filename><cr>
: Writes a file of all abbrev definitions, before dumping.

M-X Read Word Abbrev File♦<filename><cr>
: Reads file of abbrev definitions at init-time.

M-X Write Incremental Word Abbrev File♦<filename><cr>
: Writes a file of abbrev definitions changed since dumping.

M-X Read Incremental Word Abbrev File♦<filename><cr>
: Reads file of changed abbrev definitions at startup-time.

Some users with highly customized EMACS environments (their init files take a long time to run) "dump out" their environments, in effect creating another EMACS-like program (the "dump") which starts up much faster. (For instance, 1.7 cpu seconds instead of 70.5 cpu seconds. See the file INFO;MKDUMP >, for details about a simple method of dumping environments. See the file INFO;CONV >, for details about more general environment dumping.) Since the dumped environment contains word abbrev definitions, a dumped environment with hundreds of abbrevs can start just as quickly as if it had none. (But reading all these abbrevs with M-X Read Word Abbrev File in the init file originally took a long time.) For these users it is important, at dump-startup time, to read in only those abbrevs which were changed or defined *since* the environment was dumped out. A file which contains only these new abbrev's definitions is called an *incremental word abbrev file*. (It also can specify that certain abbrevs are to be killed if they were defined when the environment was dumped out, but subsequently killed.)

The startup for the dump should use the Read Incremental Word Abbrev File function instead of Read Word Abbrev File. It takes the filename as a string argument, which defaults to INCABS >. The command M-X Write Incremental Word Abbrev File♦<filename><cr> writes such a file, writing out those abbrevs more recent than the dump (ones read by Read Incremental Word Abbrev File and ones defined in the current editing session).

Setting Save Word Abbrevs to -1 will cause an incremental abbrev file to be automatically written, if necessary, when EMACS is exited.

When you want to dump out a new EMACS, first create a new, complete word abbrev definition file using M-X Write Word Abbrev File. This file now has *all* abbrevs

Chapter Twenty-Six

The PICTURE Subsystem, an Editor for Text Pictures

If you want to create a picture made out of text characters (for example, a picture of the division of a register into fields, as a comment in a program), the PICTURE package can make it easier.

Do M-X Load Lib◊PICTURE<cr>, and then M-X Edit Picture is available. Do M-X Edit Picture with point and mark surrounding the picture to be edited. Edit Picture enters a recursive editing level (which you exit with C-M-C, as usual) in which certain commands are redefined to make picture editing more convenient.

While you are inside Edit Picture, all the lines of the picture are padded out to the margin with spaces. This makes two-dimensional motion very convenient; C-B and C-F move horizontally, and C-N and C-P move vertically without the inaccuracy of a ragged right margin. When you exit from Edit Picture, spaces at the ends of lines are removed. Nothing stops you from moving outside the bounds of the picture, but if you make any changes there slightly random things may happen.

Edit Picture makes alteration of the picture convenient by redefining the way printing characters and Rubout work. Printing characters are defined to replace (overwrite) rather than inserting themselves. Rubout is defined to undo a printing character: it replaces the previous character with a space, and moves back to it.

Return is defined to move to the beginning of the next line. This makes it usable for moving to the next apparently blank (but actually filled with nothing but spaces) line, just as you use Return normally with lines that are really empty. C-O creates new blank lines after point, but they are created full of spaces.

Tab is redefined to indent (by moving over spaces, not inserting them) to under the first non-space on the previous line. Linefeed is as usual equivalent to Return followed by Tab.

Four movement-control commands exist to aid in drawing vertical or horizontal lines: If you give the command M-X Up Picture Movement, each character you type thereafter will cause the cursor to move up instead of to the right. Thus if you want to draw a line of dashes up to some point, you can give the command Up Picture Movement, type enough dashes to make the line, and then give the command Right Picture Movement to put things back to normal. Similarly, there are functions to cause downward and leftward movement: Down Picture Movement and Left Picture Movement. These commands remain in effect only until you exit the Edit Picture function. (One final note: you can use these cursor movement commands outside of

Chapter Twenty-Seven

Sorting Functions

The SORT library contains functions called Sort Lines, Sort Paragraphs and Sort Pages, to sort the region alphabetically line by line, paragraph by paragraph or page by page. For example, Sort Lines rearranges the lines in the region so that they are in alphabetical order.

Paragraphs are defined in the same way as for the paragraph-motion functions (See section 11.2 [Paragraphs], page 46.) and pages are defined as for the page motion commands (See section 18 [Pages], page 83.). All of these functions can be undone by the Undo command (See section 24.3 [Undo], page 136.). A numeric argument tells them to sort into reverse alphabetical order.

You can rearrange pages to any way you like using the functions Make Page Permutation Table and Permute Pages From Table. Make Page Permutation Table starts you editing a table containing the first line of each page. This table is kept in a buffer named *Permutation Table*. You specify the new ordering for the pages by rearranging the first lines into the desired order. You can also omit or duplicate pages by omitting or duplicating the lines.

When you are finished rearranging the lines, use Permute Pages From Table to rearrange the entire original file the same way. Reselect the original buffer first. The permuted version is constructed in a buffer named *Permuted File*. The original buffer is not changed. You can use Insert Buffer to copy the data into the original buffer.

Appendix I

Particular Types of Terminals

I.1. Ideal Keyboards

An ideal EMACS keyboard can be recognized because it has a Control key and a Meta key on each side, with another key labeled Top above them.

On an ideal keyboard, to type any character in the 9-bit character set, hold down Control or Meta as appropriate while typing the key for the rest of the character. To type C-M-K, type K while holding down Control and Meta.

You will notice that there is a key labeled "Escape" and a key labeled "Alt". The Altmode character is the one labeled "Alt". "Escape" has other functions entirely; it is handled by ITS and has nothing to do with EMACS. While we are talking about keys handled by ITS, on Meta keyboards the way to interrupt a program is CALL, rather than Control-Z, and entering communicate mode uses the BACK-NEXT key rather than Control-_. CALL *echoes* as ↑Z, but if you type C-Z it is just an ordinary character which happens to be an EMACS command to return to the superior. Similarly, BACK-NEXT echoes as ↑_ but if you type ↑_ it is just an EMACS command which happens not to be defined.

The key labeled "Top" is an extra shift key. It is used to produce the peculiar "SAIL" graphics characters which appear on the same keys as the letters. The "Shift" key gets you upper-case letters, but "Top" gets you the SAIL characters. As EMACS commands, these characters are normally self-inserting, like all printing characters. But once inserted, SAIL characters are really the same as ASCII control characters, and since characters in files are just 7 bits there is no way to tell them apart. EMACS can display them either as ASCII control characters, using an uparrow or caret to indicate them, or it can display them as SAIL characters, whichever you like. The command Control-Alpha (SAIL Character Mode) toggles the choice. Alpha is a SAIL character and you can only type it on a terminal with a Top key, but only those terminals can display the SAIL characters anyway. SAIL characters are displayed if the variable SAIL Character Mode is nonzero.

One other thing you can do with the Top key is type the Help character, which is Top-H on these keyboards. BACK-NEXT H also works, though.

For inserting an Altmode, on an ideal keyboard you can type C-M-Altmode. C-Altmode is a synonym for C-M-C (^R Exit).

The "bit prefix" characters that you must use on other terminals are also available on terminals with Meta keys, in case you find them more convenient or get into habits on those other terminals.

including a few that Altmode can't be used for because the corresponding non-Meta character isn't on the keyboard. Thus, while you can't type C M-; as Altmode Control-;, since there is no Control-; in ASCII, you can type C-M-; as C-C ;. The Control (non-Meta) characters which can't be typed directly require the use of C-^, as in C-^ < to get the effect of C-<. Because C-^ by itself is hard to type, the EMACS command set is arranged so that most of these non-ASCII Control characters are not very important. Usually they have synonyms which are easier to type. In fact, in this manual only the easier-to-type forms are usually mentioned.

In order to type the Help character you must actually type two characters, C-_ and H. C-_ is an escape character for ITS itself, and C-_ followed by H causes ITS to give the Help character as input to EMACS.

On ASCII keyboards, you can type a numeric argument by typing an Altmode followed by the minus sign and/or digits. Then comes the command for which the argument is intended. For example, type Altmode 5 C-N to move down five lines. If the command is a Meta command, it must have an Altmode of its own, as in Altmode 5 Altmode F to move forward five words.

Note to customizers: this effect requires redefining the Meta-digit commands, since the Altmode and the first digit amount to a Meta-digit character. The new definition is ^R Autoarg, and the redefinition is done by the default init file.

If you use numeric arguments very often, and you dislike having to start one with an Altmode, you might enjoy using Autoarg mode, in which you can specify a numeric argument by just typing the digits. See section 4 [Arguments], page 17, for details.

I.4. Upper-case-only Terminals

On terminals lacking the ability to display or enter lower case characters, a special input and output case-flagging convention has been defined for editing files which contain lower case characters.

The customary escape convention is that a slash prefixes any upper case letter; all unprefixed letters are lower case (but see below for the "lower case punctuation characters"). This convention is chosen because lower case is usually more frequent in files containing any lower case at all. Upper case letters are displayed with a slash ("/") in front. Typing a slash followed by a letter is a good way to insert an upper case letter. Typing a letter without a slash inserts a lower case letter. For the most part, the buffer will appear as if the slashes had simply been inserted (type /A and it inserts an upper case A, which displays as /A), but cursor-motion commands will reveal that the slash and the A are really just one character. Another way to insert an upper-case letter is to quote it with C-Q.

Note that this escape convention applies only to display of the buffer and insertion in the buffer. It does not apply to arguments of commands (it is hardly ever useful for them, since case is ignored in command names and most commands' arguments). Case conversion is performed when you type commands into the minibuffer, but not when the commands are actually executed.

The ASCII character set includes several punctuation characters whose codes fall

I.5.2. SLOWLY Commands

The commands provided are:

M-O (^R Set Screen Size)
: This function reduces the amount of the screen used for displaying your text, down to a few lines at the top or the bottom. If called without an argument, it will use the same size as last time (or 3 if it hasn't been called before). If given a positive argument, that is taken to be the number of lines to use at the top of the screen. If given a negative argument, it is taken to be the number of lines at the bottom of the screen. If given an argument of 0, it returns to the use of the entire screen. The section of the screen that is in use is (defaultly) delimited by a line of 6 dashes. This command sets the variable Short Display Size.

C-S (^R Slow Display I-Search)
: This function is just like the usual incremental search, except if the search would run off the screen and cause a redisplay, it narrows the screen to use only a few lines at the top or bottom of the screen to do the redisplay in. When the search is exited, use of the full screen resumes. The size of the window used for the search is the value of the variable Slow Search Lines. If it is positive, it is the number of lines at top of screen; if negative, it is the number of lines at bottom of screen. The default is 1. The variable Slow Search Separator contains the string used to show the end of the search window. By default it is six dashes. See section 10 [Search], page 41.

C-R (^R Slow Reverse Display I-Search)
: This searches in backwards in the style of ^R Slow Display I-Search.

C-X Q (^R Edit Quietly)
: This function enters a recursive editing level with redisplay inhibited. This means that your commands are carried out but the screen does not change. C-L with no argument redisplays. So you can update the screen when you want to. Two C-L's in a row clear the screen and redisplay. C-L with an argument repositions the window, as usual (See section 15 [C-L], page 75.). To exit and resume continuous redisplay, use C-M-C.

I.5.3. Minibuffers

SLOWLY provides control over how minibuffers display on your screen. The variable Minibuffer Size specifies how many lines it takes up. If this is made negative, the minibuffer will appear at the bottom of the screen instead of the top. Thus one mode of operation which some people like is to use ^R Set Screen Size to set up to not use the bottom 3 lines of the screen, and set Minibuffer Size to -3. This will permanently reserve 3 lines at the bottom of the screen for the minibuffer. See section 23 [Minibuffer], page 131.

The variable Minibuffer Separator holds the string used to separate the minibuffer area from the rest of the screen. By default, this is six dashes.

Appendix II

Use of EMACS from Printing Terminals

While EMACS was designed to be used from a display terminal, you can use it effectively from a printing terminal. You cannot, however, learn EMACS using one.

All EMACS commands have the same editing effect from a printing terminal as they do from a display. All that is different is how they try to show what they have done. EMACS attempts to make the same commands that you would use on a display terminal act like an interactive line-editor. It does not do as good a job as editors designed originally for that purpose, but it succeeds well enough to keep you informed of what your commands are accomplishing, provided you know what they are supposed to do and know how they would look on a display.

The usual buffer display convention for EMACS on a printing terminal is that the part of the current line before the cursor is printed out, with the cursor following (at the right position in the line). What follows the cursor on the line is not immediately visible, but normally you will have a printout of the original contents of the line a little ways back up the paper. For example, if the current line contains the word "FOOBAR", and the cursor is after the "FOO", just "FOO" would appear on the paper, with the cursor following it. Typing the C-F command to move over the "B" would cause "B" to be printed, so that you would now see "FOOB" with the cursor following it. All forward-motion commands that move reasonably short distances print out what they move over.

Backward motion is handled in a complicated way. As you move back, the terminal backspaces to the correct place. When you stop moving back and do something else, a linefeed is printed first thing so that the printing done to reflect subsequent commands does not overwrite the text you moved back over and become garbled by it. The Rubout command acts like backward motion, but also prints a slash over the character rubbed out. Other backwards deletion commands act like backward motion; they do not print slashes (it would be an improvement if they did).

One command is different on a printing terminal: C-L, which normally means "clear the screen and redisplay". With no argument, it retypes the entire current line. An argument tells it to retype the specified number of lines around the current line.

On printing terminals, C-S (^R Incremental Search) does not print out the context automatically. To see what you have found at any stage, type C-L. This types out the current line but does not exit the search. All the normal facilities of incremental searching are available for finding something else if you had not found the right place initially.

Distribution of EMACS

EMACS is available for distribution for use on Tenex and Twenex systems. To get it, mail me a 2400 foot mag tape with a self-addressed return mailing envelope. It should hold both the tape and a manual.

EMACS does not cost anything; instead, you are joining the EMACS software-sharing commune. The conditions of membership are that you must send back any improvements you make to EMACS, including any libraries you write, and that you must not redistribute the system except exactly as you got it, complete. (You can also distribute your customizations, *separately*.)

Please do not attempt to get a copy of EMACS, for yourself or any one else, by dumping it off of your local system. It is almost certain to be incomplete or inconsistent. It is pathetic to hear from sites that received incomplete copies lacking the sources, asking me years later whether sources are available. (All sources are distributed, and should be on line at every site so that users can read them and copy code from them). If you wish to give away a copy of EMACS, copy a distribution tape from MIT, or mail me a tape and get a new one.

EMACS does not run on Bottoms-10; conversion would be painful but possible. Nor does it run on any computers except the PDP-10. However, there are several other implementations of EMACS for other systems. There are also several ersatz EMACSes, which are editors that superficially resemble EMACS but lack the extensibility which is the essential feature of EMACS. Here is a list of those that run on systems in general use, and how to obtain them.

- MULTICS EMACS. This true EMACS, written in Lisp, is a Honeywell product and runs on Multics systems only. Unfortunately, it costs an arm and a leg. An early version was distributed free to Multics sites; perhaps your Multics site can get this from another one.

- NILE. This true EMACS, written in New Implementation of Lisp, will run on VAXes under VMS and UNIX when it is available, perhaps next summer. Write to Richard Soley; Lab for Computer Science; 545 Tech Square; Cambridge, MA 02139.

- PRIME EMACS. This true EMACS, containing an implementation of Lisp, will at some time be available from PRIME itself. Write to Barry Kingsbury; 10B-7; Prime Computer Company; 500 Old Connecticut Path; Framingham, MA 01701.

- VAX EMACS. This is a semi-ersatz EMACS, containing a Lisp-like extension language which currently lacks the data types required for general programming. It runs under VMS and UNIX. Write to James

Glossary

Aborting
: Aborting a recursive editing level (q.v.) means canceling the command which invoked the recursive editing. For example, if you abort editing a message to be sent, the message is not sent. Aborting is done with the command C-]. See section 24.1 [Aborting], page 133.

Altmode
: Altmode is a character, labelled Escape on some keyboards. It is the bit prefix character (q.v.) used to enter Meta-characters when the keyboard does not have a Meta (q.v.) key. See section 2 [Characters], page 9. Also, it delimits string arguments to extended commands. See section 5 [Extended Commands], page 19.

Balance Parentheses
: EMACS can balance parentheses manually or automatically. You can ask to move from one parenthesis to the matching one. See section 20.6.1 [Lists], page 97. When you insert a close parenthesis, EMACS can show the matching open. See section 20.4 [Matching], page 94.

Bit Prefix Character
: A bit prefix character is a command which combines with the next character typed to make one character. They are used for effectively typing commands which the keyboard being used is not able to send. For example, to use a Meta-character when there is no Meta key on the keyboard, the bit prefix character Altmode (q.v.) is needed. See section 2 [Characters], page 9.

Buffer
: The buffer is the basic editing unit; one buffer corresponds to one piece of text being edited. You can have several buffers, but at any time you are editing only one, the "selected" buffer, though two can be visible when you are using two windows. See section 14 [Buffers], page 71.

C-
: C is an abbreviation for Control, in the name of a character. See section 2 [Characters], page 9.

C-M-
: C-M- is an abbreviation for Control-Meta, in the name of a character. See section 2 [Characters], page 9.

Comment
: A comment is text in a program which is intended only for humans reading the program, and is marked specially so that the compiler will ignore it. EMACS offers special commands for creating and killing comments. See section 20.5 [Comments], page 95.

Command
: A command is a character or sequence of characters which, when typed by the user, fully specifies one action to be performed by EMACS. For example, "X" and "Control-F" and "Meta-X Text

It is so named because most such lists are calls to the Lisp function defun. See section 20.6.2 [Defuns], page 100.

Delete — This is the label used on some keyboards for the Rubout character.

Deletion — Deletion means erasing text without saving it. EMACS deletes text only when it is expected not to be worth saving (all whitespace, or only one character). The alternative is killing (q.v.). See section 9.1 [Killing], page 35.

Dispatch Table — The dispatch table is what records the connections (q.v.) from command characters to functions. Think of a telephone switchboard connecting incoming lines (commands) to telephones (functions). A standard EMACS has one set of connections; a customized EMACS may have different connections. See section 5.2 [Functions], page 21.

Echo Area — The echo area is the bottom three lines of the screen, used for echoing the arguments to commands, for asking questions, and printing brief messages. See section 1 [Screen], page 5.

Echoing — Echoing is acknowledging the receipt of commands by displaying them (in the echo area). Most programs other than EMACS echo all their commands. EMACS never echoes single-character commands; longer commands echo only if you pause while typing them.

Error Messages — Error messages are single lines of output printed by EMACS when the user or a TECO program asks for something impossible. They appear at the top of the screen and end with a question mark.

Escape — Escape is the label used on some keyboards for the Altmode character.

Exiting — Exiting EMACS means returning to EMACS's superior, normally HACTRN. See section 6.4 [Exiting], page 27. Exiting a recursive editing level (q.v.) means allowing the command which invoked the recursive editing to complete normally. For example, if you are editing a message to be sent, and you exit, the message is sent.

Extended Command — An extended command is a command which consists of the character Meta-X followed by the command name (really, the name of a function (q.v.)). An extended command requires several characters of input, but its name is made up of English words, so it is easy to remember. See section 5 [Extended Commands], page 19.

Extension — Extension means making changes to EMACS which go beyond the bounds of mere customization. If customization is moving the furniture around in a room, extension is building new furniture. See the file INFO;CONV >.

Filling — Filling text means moving text from line to line so that all the lines are approximately the same length. See section 11.4 [Filling], page 50.

Function — A function is a named subroutine of EMACS. When you type a

commands", and an individual extended command is often referred to as "M-X such-and such". See section 5 [M-X], page 19.

Major Mode — The major modes are a mutually exclusive set of options which configure EMACS for editing a certain sort of text. Ideally, each programming language has its own major mode. See section 20.1 [Major Modes], page 91.

Mark — The mark points, invisibly, to a position in the text. Many commands operate on the text between point and the mark (known as the region, q.v.). See section 8 [Mark], page 31.

Meta — Meta refers to the Meta key. A character's name includes the word Meta if the Meta key must be held down in order to type the character. If there is no Meta key, then the Altmode character is used as a prefix instead. See section 2 [Characters], page 9.

Meta Character — A Meta character is one whose character code includes the Meta bit. These characters can be typed only by means of a Meta key or by means of the metizer command (q.v.).

Metizer — The metizer is another term for the bit prefix character for the Meta bit; namely, Altmode (q.v.).

Minibuffer — The minibuffer is a facility for editing and then executing a TECO program. See section 23 [Minibuffer], page 131.

Minor mode — A minor mode is an optional feature of EMACS which can be switched on or off independently of all other features. Each minor mode is both the name of an option (q.v.) and the name of an extended command to set the option. See section 22.1 [Minor Modes], page 115.

MM-command — This is an obsolete synonym for "extended command".

Mode line — The mode line is a line just above the echo area (q.v.), used for status information. See section 1.1 [Mode Line], page 6.

Narrowing — Narrowing means limiting editing to only a part of the text in the buffer. Text outside that part is inaccessible to the user until the boundaries are widened again, but it is still there, and saving the file saves it all. See section 17 [Narrowing], page 81.

Node — The node is the unit of structure of INFO (q.v.) files. When referring to documentation contained only in INFO files, we sometimes refer to a node of a specific name, in a specific file, as in "See the file INFO;CONV >, node Hooks".

Numeric Argument — A numeric argument is a number specified before a command to change the effect of the command. Often the numeric argument serves as a repeat count. See section 4 [Numeric Arguments], page 17.

Option — An option is a variable which exists to be set by the user to change the behavior of EMACS commands. This is an important method of customization. See section 22.3 [Variables], page 118.

Parse — We say that EMACS parses words or expressions in the text being

Rubout
: Rubout is a character, sometimes labelled "Delete ' It is used as a command to delete one character of text. It also deletes one character when an EMACS command is reading an argument.

S-expression
: An s-expression is the basic syntactic unit of Lisp: either a list, or a symbol containing no parentheses (actually, there are a few exceptions to the rule, based on the syntax of Lisp). See section 20.6.1 [Lists], page 97.

Selecting
: Selecting a buffer (q.v.) means making editing commands apply to that buffer as opposed to any other. At all times one buffer is selected and editing takes place in that buffer. See section 14 [Buffers], page 71.

Self-documentation
: Self-documentation is the feature of EMACS which can tell you what any command does, or give you a list of all commands related to a topic you specify. You ask for self-documentation with the Help character. See section 7 [Help], page 29.

String Argument
: A string argument is an argument which follows the command name in an extended command. In "M-X Apropos♦word<cr>", "Word" is a string argument to the Apropos command. See section 5 [Extended Commands], page 19.

Subsystem
: A subsystem of EMACS is an EMACS command which, itself, reads commands and displays the results. Examples are INFO, which is for perusing documentation; DIRED, which is for editing directories; RMAIL and BABYL, which are for reading and editing mail. The word "subsystem" implies that it offers many independent commands which can be used freely. If an EMACS function asks specific questions, we do not call it a subsystem.

  Usually the subsystem continues in operation until a specific command to exit (usually "Q") is typed. The commands for a subsystem do not usually resemble ordinary EMACS commands, since editing text is not their purpose. The Help character should elicit the subsystem's documentation. See section 6.1 [Subsystems], page 25.

Syntax Table
: The syntax table tells EMACS which characters are part of a word, which characters balance each other like parentheses, etc. See section 22.4 [Syntax], page 119.

Tailoring
: This is a synonym for customization (q.v.).

TECO Search String
: A TECO search string is a sort of pattern used by the TECO search command, and also by various EMACS commands which use the TECO search command. See section 19.3 [TECO search strings], page 89.

Top Level
: Top level is the normal state of EMACS, in which you are editing the text of the file you have visited. You are at top level whenever you are not in a recursive editing level or a subsystem (q.v.).

Twenex
: Twenex is the operating system which DEC likes to call "TOPS-20". However, a person should not be forced to call a

Yanking
: This is a synonym for un-killing (q.v.).

^R
: The string "^R" is the beginning of many function names. See section 5.2 [Functions], page 21.

^R Mode
: ^R mode is the real time editing mode of TECO. EMACS *always* operates in this mode.

Command Summary

This summary contains brief descriptions with cross references for all commands, grouped by topic. Within each topic, they are in alphabetical order. Our version of alphabetical order places non-control non-meta characters first, then control characters, then meta characters, then control-meta characters. Control-X and Meta-X commands come last. Not all Meta-X commands are included.

Prefix Characters

Altmode (^R Prefix Meta)
: Altmode is a bit prefix character which turns on the Meta bit in the next character. Thus, Altmode F is equivalent to the single character Meta-F, which is useful if your keyboard has no Meta key. See section 2 [Characters], page 9.

Control-^ (^R Prefix Control)
: Control-^ is a bit prefix character which turns on the Control bit in the following character. Thus, Control-^ < is equivalent to the single character Control-<. See section 2 [Characters], page 9.

Control-C (^R Prefix Control-Meta)
: Control-C is a bit prefix character which turns on the Control bit and the Meta bit in the following character. Thus, Control-C ; is equivalent to the single character Control-Meta-;. See section 2 [Characters], page 9.

Control-Q (^R Quoted Insert)
: Control-Q inserts the following character. This is a way of inserting control characters. See section 3 [Basic Editing], page 13.

Control-U (^R Universal Argument)
: Control-U is a prefix for numeric arguments which works the same on all terminals. See section 4 [Arguments], page 17.

Control-X
: Control-X is a prefix character which begins a two-character command. Each combination of Control-X and another character is a "Control-X command". Individual Control-X commands appear in this summary according to their uses.

Meta-X (^R Extended Command)
: Meta-X is a prefix character which introduces an extended command name. See section 5 [Meta-X], page 19.

Control-Meta-X (^R Instant Extended Command)

Control-X Control-N (^R Set Goal Column)
Control-X Control-N sets a horizontal goal for the Control-N and Control-P commands. When there is a goal, those commands try to move to the goal column instead of straight up or down.

Lines

Return (^R CRLF)
Return inserts a line separator, or advances onto a following blank line. See section 3 [Basic Editing], page 13.

Control-O (^R Open Line, built-in function)
Control-O inserts a line separator, but point stays before it. See section 3 [Basic Editing], page 13.

Meta-= (^R Count Lines Region)
Meta-= prints the number of lines between point and mark. See section 8 [Mark], page 31.

Control-X Control-O (^R Delete Blank Lines)
Control-X Control-O deletes all but one of the blank lines around point. If the current line is not blank, all blank lines following it are deleted. See section 3 [Basic Editing], page 13.

Control-X Control-T (^R Transpose Lines)
Control-X Control-T transposes the contents of two lines. See section 12 [Fixing Typos], page 57.

Killing and Un-killing

Rubout (^R Backward Delete Character, built-in function)
Rubout deletes the previous character. See section 3 [Basic Editing], page 13.

Control-Rubout (^R Backward Delete Hacking Tabs, built-in function)
Control-Rubout deletes the previous character, but converts a tab character into several spaces. See section 20.6 [Lisp], page 97.

Control-D (^R Delete Character, built-in function)
Control-D deletes the next character. See section 3 [Basic Editing], page 13.

Control-K (^R Kill Line)
Control-K kills to the end of the line, or, at the end of a line, kills the line separator. See section 9.1 [Killing], page 35.

Control-W (^R Kill Region)
Control-W kills the region, the text between point and the mark. See section 9.1 [Killing], page 35. See section 8 [Region], page 31.

Control-Y (^R Un-kill)
Control-Y reinserts the last saved block of killed text. See section 9.2 [Un-Killing], page 37.

Control-Meta-V (^R Scroll Other Window)
: Control-Meta-V scrolls the other window up or down, when you are in two window mode. See section 16 [Windows], page 77.

M-X View Buffer
: M-X View Buffer skips through a buffer by screenfuls. See section 15 [Display], page 75.

M-X View File
: M-X View File lets you move through a file sequentially by screenfuls forward and back. See section 13.7 [View File], page 68.

The Mark and the Region

Control-Space (^R Set/Pop Mark)
: Control-Space sets the mark or moves to the location of the mark. See section 8 [Mark], page 31.

Control-< (^R Mark Beginning)
: Control-< sets the mark at the beginning of the buffer. See section 8 [Mark], page 31.

Control-> (^R Mark End)
: Control-> sets the mark at the end of the buffer. See section 8 [Mark], page 31.

Control-@ (^R Set/Pop Mark)
: Control-@ sets the mark or moves to the location of the mark. See section 8 [Mark], page 31.

Meta-@ (^R Mark Word)
: Meta-@ puts the mark at the end of the next word. See section 11.1 [Words], page 45.

Meta-H (^R Mark Paragraph)
: Meta-H puts point at the beginning of the paragraph and the mark at the end. See section 11.2 [Sentences], page 46.

Control-Meta-@ (^R Mark Sexp)
: Control-Meta-@ puts the mark at the end of the next s-expression. See section 20.6.1 [Lists], page 97.

Control-Meta-H (^R Mark Defun)
: Control-Meta-H puts point at the beginning of the current Defun and the mark at the end. See section 20.6.2 [Defuns], page 100.

Control-X H (^R Mark Whole Buffer)
: Control-X H puts point at the beginning of the buffer and the mark at the end. See section 8 [Mark], page 31.

Control-X Control-P (^R Mark Page)
: Control-X Control-P puts point at the beginning of the current page and the mark at the end. See section 18 [Pages], page 83.

Control-X Control-X (^R Exchange Point and Mark)
: Control-X Control-X sets point where the mark was and the mark where point was. See section 8 [Mark], page 31.

M-X Indent Tabs Mode
: M-X Indent Tabs Mode turns Indent Tabs mode on or off. When Indent Tabs mode is on, the indentation commands use tab characters for indentation whenever possible. Otherwise they use only spaces. See section 22.1 [Minor Modes], page 115.

M-X Tabify
: M-X Tabify converts spaces after point to tabs when that can be done without changing the appearance. See section 11.3 [Indenting Text], page 48.

M-X Untabify
: M-X Untabify converts all tabs after point to spaces. A numeric argument says how far apart the tab stops are, which is good for converting files brought from systems with tab stops at intervals other than 8. See section 11.3 [Indenting Text], page 48.

Words, Sentences and Paragraphs

Meta-A (^R Backward Sentence)
: Meta-A moves to the beginning of the sentence. See section 11.2 [Sentences], page 46.

Meta-B (^R Backward Word)
: Meta-B moves backward one word. See section 11.1 [Words], page 45.

Meta-D (^R Kill Word)
: Meta-D kills one word forward. See section 11.1 [Words], page 45.

Meta-E (^R Forward Sentence)
: Meta-E moves to the end of the sentence. See section 11.2 [Sentences], page 46.

Meta-F (^R Forward Word)
: Meta-F moves forward one word. See section 11.1 [Words], page 45.

Meta-H (^R Mark Paragraph)
: Meta-H puts point at the front of the current paragraph and the mark at the end. See section 11.2 [Sentences], page 46.

Meta-K (^R Kill Sentence)
: Meta-K kills to the end of the sentence. See section 11.2 [Sentences], page 46.

Meta-T (^R Transpose Words)
: Meta-T transposes two consecutive words. See section 11.1 [Words], page 45.

Meta-[(^R Backward Paragraph)
: Meta-[moves to the beginning of the paragraph. See section 11.2 [Sentences], page 46.

Meta-] (^R Forward Paragraph)

Control-] (Abort Recursive Edit)
: Control-] aborts a recursive editing level; that is to say, exits it without allowing the command which invoked it to finish. See section 24.1 [Quitting], page 133.

Control-Meta-C (^R Exit, built-in function)
: Control-Meta-C exits from a recursive editing level and allows the command which invoked the recursive editing level to finish. At top level, it exits from EMACS to its superior job. See section 6.4 [Exiting], page 27.

Control-X Control-C (^R Return to Superior)
: Control-X Control-C returns from EMACS to its superior job, even if EMACS is currently inside a recursive editing level. In that case, re-entering EMACS will find it still within the recursive editing level. See section 6.4 [Exiting], page 27.

M-X Compile
: M-X Compile recompiles the file you are visiting, in a manner that depends on the major mode. See section 20.2 [Compile], page 92.

M-X Top Level
: M-X Top Level returns to the top level EMACS command loop or to TECO. See section 24.1 [Quitting], page 133.

M-X Undo
: M-X Undo retracts the last undoable change to the buffer. See section 24.2 [Lossage], page 134.

Pages

Control-X L (^R Count Lines Page)
: Control-X L prints the number of lines on the current page, and how many come before point and how many come after. See section 18 [Pages], page 83.

Control-X P (^R Narrow Bounds to Page)
: Control-X P narrows the virtual boundaries to the current page. See section 17 [Narrowing], page 81.

Control-X [(^R Previous Page)
: Control-X [moves backward to the previous page boundary. See section 18 [Pages], page 83.

Control-X] (^R Next Page)
: Control-X] moves forward to the next page boundary. See section 18 [Pages], page 83.

Control-X Control-P (^R Mark Page)
: Control-X Control-P puts point at the beginning and the mark at the end of the current page. See section 18 [Pages], page 83.

M-X View Page Directory (in PAGE)
: M-X View Page Directory prints a directory of the pages of the file. See section 18.1 [PAGE], page 84.

Control-Meta-K (^R Kill Sexp)
: Control-Meta-K kills the following s-expression. See section 20.6.1 [Lists], page 97.

Control-Meta-N (^R Next List)
: Control-Meta-N moves forward over one list, ignoring atoms before the first open parenthesis. See section 20.6.1 [Lists], page 97.

Control-Meta-P (^R Previous List)
: Control-Meta-P moves backward over one list, ignoring atoms reached before the first close parenthesis. See section 20.6.1 [Lists], page 97.

Control-Meta-Q (^R Indent Sexp)
: Control-Meta-Q adjusts the indentation of each of the lines in the following s-expression, but not the current line. See section 20.3 [Indenting], page 93.

Control-Meta-T (^R Transpose Sexps)
: Control-Meta-T transposes two consecutive s-expressions. See section 20.6.1 [Lists], page 97.

Control-Meta-U (^R Backward Up List)
: Control-Meta-U moves backward up one level of list structure. See section 20.6.1 [Lists], page 97.

Files

Meta-. (^R Find Tag)
: Meta-. moves to the definition of a specific function, switching files if necessary. See section 21 [TAGS], page 107.

Meta-~ (^R Buffer Not Modified)
: Meta-~ clears the flag which says that the buffer contains changes that have not been saved. See section 13.1 [Visiting], page 61.

Control-X Control-F (Find File)
: Control-X Control-F visits a file in its own buffer. See section 14 [Buffers], page 71.

Control-X Control-Q (^R Set File Read Only)
: Control-X Control-Q makes the visited fie read only, or no longer read only. See section 13.1 [Visiting], page 61.

Control-X Control-S (^R Save File)
: Control-X Control-S saves the visited file. See section 13.1 [Visiting], page 61.

Control-X Control-V (^R Visit File)
: Control-X Control-V visits a file. See section 13.1 [Visiting], page 61.

Control-X Control-W (Write File)
: Control-X Control-W saves the file, asking for names to save it under. See section 13.7 [Advanced File Commands], page 68.

M-X Clean Directory
: M-X Clean Directory deletes all but the most recent versions of every file in a directory. See section 13.5 [Cleaning Directories], page 66.

M-X Compare Directories
: M-X Compare Directories compare two directories with the same name on different machines. See section 13.8 [Compare Directories], page 69.

M-X List Directories
: M-X List Directories list the names of all disk directories. See section 13.4 [Directories], page 65.

M-X List Files
: M-X List Files prints a very brief listing of a directory, listing only the filenames, several files per line. See section 13.4 [Directories], page 65.

M-X Reap File
: M-X Reap File deletes all but the most recent versions of a file. See section 13.5 [Cleaning Directories], page 66.

M-X View Directory
: M-X View Directory prints a file directory. See section 13.4 [Directories], page 65.

Buffers

Control-X Control-B (List Buffers)
: Control-X Control-B prints a list of all buffers, their major modes and the files they are visiting. See section 14 [Buffers], page 71.

Control-X A (^R Append to Buffer)
: Control-X A adds the text of region into another buffer. See section 9.3 [Copying], page 39.

Control-X B (Select Buffer)
: Control-X B is the command for switching to another buffer. See section 14 [Buffers], page 71.

Control-X K (Kill Buffer)
: Control-X K kills a buffer. See section 14 [Buffers], page 71.

M-X Insert Buffer
: M-X Insert Buffer inserts the contents of another buffer into the existing text of this buffer. See section 14 [Buffers], page 71.

M-X Kill Some Buffers
: M-X Kill Some Buffers offers to kill each buffer. See section 14 [Buffers], page 71.

M-X Make Space
: M-X Make Space tries to free up space inside EMACS for more libraries or buffers. See section 24.2.5 [Storage Exhausted], page 135.

Control-X Control-L (^R Lowercase Region)
Control-X Control-L converts the text of the region to lower case. See section 11.5 [Case], page 51.

Control-X Control-U (^R Uppercase Region)
Control-X Control-U converts the text of the region to upper case. See section 11.5 [Case], page 51.

Minor Corrections

Meta-# (^R Change Font Region)
Meta-# inserts a font-change command good for certain text justifiers around a word. See section 11.6 [Fonts], page 52.

Meta-$ (^R Correct Word Spelling)
Meta-$ (Dollar sign, not Altmode) passes the word before point to the SPELL program. If it is not a correct spelling, you have the option of replacing it with a corrected spelling. See section 12 [Fixing Typos], page 57.

Meta-' (^R Upcase Digit)
Meta-' converts a digit before point on the same or previous line to a punctuation character, assuming that you failed to type the shift key and thus typed the digit by mistake. See section 12 [Fixing Typos], page 57.

Meta-_ (^R Underline Word)
Meta-_ inserts underlining commands good for certain text justifiers around a word. See section 11.7 [Underlining], page 53.

Control-X # (^R Change Font Region)
Control-X # inserts font change commands good for certain text justifiers around the region. See section 11.6 [Fonts], page 52.

Control-X _ (^R Underline Region)
Control-X _ inserts underlining commands good for certain text justifiers around the region. See section 11.7 [Underlining], page 53.

Windows

Control-Meta-V (^R Scroll Other Window)
Control-Meta-V scrolls the other window up or down. See section 15 [Display], page 75.

Control-X 1 (^R One Window)
Control-X 1 returns to one-window mode. See section 16 [Windows], page 77.

Control-X 2 (^R Two Windows)
Control-X 2 splits the screen into two windows. See section 16 [Windows], page 77.

M-X List Variables lists the names and values of all variables, or of those whose names contain a specified string. See section 22.3 [Variables], page 118.

M-X List Redefinitions

M-X List Redefinitions describes all the ways which the major mode and local modes of the selected buffer modify the standard EMACS. See section 20.1 [Major Modes], page 91.

M-X What Page

M-X What Page prints the page and line number of point. See section 18 [Pages], page 83.

Keyboard Macros

Control-X ((^R Start Kbd Macro)

Control-X (begins defining a keyboard macro. See section 22.8 [KBDMAC], page 128.

Control-X) (^R End Kbd Macro)

Control-X) terminates the definition of a keyboard macro. See section 22.8 [KBDMAC], page 128.

Control-X E (^R Call Last Kbd Macro)

Control-X E executes the most recently defined keyboard macro. See section 22.8 [KBDMAC], page 128.

Control-X Q (^R Kbd Macro Query)

Control-X Q in a keyboard macro can ask the user whether to continue or allow him to do some editing before continuing with the keyboard macro. See section 22.8 [KBDMAC], page 128.

M-X Name Kbd Macro

M-X Name Kbd Macro gives a permanent name to the last keyboard macro defined. See section 22.8 [KBDMAC], page 128.

M-X View Kbd Macro

M-X View Kbd Macro prints the definition of a keyboard macro. See section 22.8 [KBDMAC], page 128.

Libraries

M-X Kill Libraries

M-X Kill Libraries discards one or more libraries from core. See section 22.2 [Libraries], page 116.

M-X List Library

M-X List Library describes briefly all the functions in a library. See section 22.2 [Libraries], page 116.

M-X Load Library

M-X Load Library loads one library, permanently. See section 22.2 [Libraries], page 116.

M-X View Mail
: M-X View Mail displays your own or another user's mail file using View File. See section 6.2 [Mail], page 26.

Minibuffer

Control-% (^R Replace String)
: Control-% invokes a minibuffer containing a call to Replace String. You fill in the arguments. See section 19 [Replace], page 87.

Meta-Altmode (^R Execute Minibuffer)
: Meta-Altmode invokes an empty minibuffer which you can fill in with a TECO program to be executed. See section 23 [Minibuffer], page 131.

Meta-% (^R Query Replace)
: Meta-% invokes a minibuffer containing a call to Query Replace. You fill in the arguments. See section 19 [Replace], page 87.

Control-X Altmode (^R Re-execute Minibuffer)
: Control-X Altmode re-executes a TECO program previously executed in the minibuffer. It can also re-execute an extended command. See section 23 [Minibuffer], page 131.

Catalog of Libraries

Libraries Used Explicitly

These are libraries which you must load with M-X Load Library↑<libname><cr> to use. If no cross-reference is given, the only documentation for the library is the self-documentation contained in it. Use M-X List Library↑<libname><cr> to print a brief description of each function in the library. For more detailed information, load the library and use M-X Describe on individual functions.

ABSTR	contains commands for making documentation files: wall charts, and abstracts of libraries. See the file INFO;CONV >.
BABYL	is a subsystem for reading, sending and editing mail. See the file INFO;BABYL >.
BACKQ	provides a feature for Maclisp, similar to automatic display of matching parentheses: when you insert a comma or atsign, the cursor moves momentarily to the backquote which dominates it.
BSHACK	has functions for operating on lines containing overprinting.
BUGHUNT	contains commands for putting your name into each comment you edit. This is to record who changed what.
CACHE	implements a cache for speeding up EMACS subroutine calls.
CHESS	implements commands for editing pictures of chess boards.
CLU	implements CLU mode. See the file INFO;ECLU >.
COLUMNS	implements commands for converting single-column text into double-column text and vice versa.
COMPLT	provides completion for buffer names and variable names.
DELIM	implements commands for moving over balanced groupings of various kinds of parentheses. There are a pair of commands for square brackets, a pair for angle brackets, etc.
DOCLSP	prints documentation from the MacLisp manual on a specified Lisp function.
DOCIX	constructs indexes of Lisp manual files for DOCLSP.
DOCOND	is a macro processor and conditionalizer for text files, useful for maintaining multiple versions of documents with one source.
DOCTOR	contains DOCTOR mode, a psychiatrist.
DRAW	offers functions for editing pictures made of characters. These partially duplicate the facilities of the PICTURE library, but contain other distinct features.

OUTLINE	implements Outline Mode, for editing outlines.
PAGE	defines commands for viewing only one page of the file at a time. See section 18.1 [PAGE], page 84.
PASCAL	implements PASCAL mode. See the file INFO;EPASC >.
PERSONAL	has functions for keeping notes on your current projects.
PHRASE	has commands for moving over and killing phrases of text.
PICTURE	contains Edit Picture, the command for editing text pictures. See section 26 [PICTURE], page 151.
PURIFY	generates libraries from EMACS source files, and contains other functions useful for editing the source files. See the file INFO; CONV >.
QSEND	sends a message to another logged-in user, like :QSEND.
RENUM	renumbers figures, equations, theorems or chapters.
RMAIL	is for reading, editing and sending mail. See the file INFO; RMAIL >.
RUNOFF	is for text-justified documents divided into separate source files. It rejustifies the files which have changed, then runs :@ to print only the pages which have changed.
SAIL	implements SAIL mode.
SCHEME	implements SCHEME mode.
SCRLIN	contains alternative definitions of C-N and C-P which move by screen lines instead of by real lines.
SLOWLY	redefines commands and options to suit slow terminals.
SORT	implements the sorting commands.
SPLIT	contains the commands Split File and Unsplit File for breaking up large files into subfiles small enough to be edited. See section 24.2 [Split], page 134.
TDEBUG	is a debugger for TECO programs. It displays the buffer in one window and the program in the other, while stepping by lines or setting breakpoints. See the file INFO;TDEBUG >.
TIME	causes the current time of day to be displayed in the mode line.
TMACS	contains miscellaneous useful functions
VT100	defines the arrow keys and numeric keypad of the VT-100 terminal to move the cursor and supply numeric arguments.
VT52	defines the numeric keypad of the VT-52 terminal to supply numeric arguments.
XLISP	contains functions for global stylistic transformations of Lisp code. See section 20.6 [Lisp], page 97.

Index of Variables

An option is a variable whose value Edit Options offers for editing. A hook variable is a variable which is normally not defined, but which you can define if you wish for customization. Most hook variables require TECO programs as their values.

The default value of the variable is given in parentheses after its name. If no value is given, the default value is zero. If the word "nonexistent" appears, then the variable does not exist unless you create it.

Abort Resumption Message
: This is the message to be printed by C-] to tell you how to resume the aborted command. If this variable is zero, there is no way to resume, so C-] asks for confirmation. See section 24.1 [Quitting], page 133.

Additional Abbrev Expanders (nonexistent)
: If this variable exists when Word Abbrev mode is turned on, it is a string of characters which should terminate and expand an abbrev, in addition to the punctuation characters which normally do so. See also WORDAB Ins Chars.

After Compilation Hook (nonexistent)
: If this variable exists and is nonzero, then it is executed as a TECO expression by the function Compile, after compilation itself is finished. Exception: If the variable Compile Command is also nonzero, it overrides this hook. See section 20.2 [Compile], page 92.

Atom Word Mode The minor mode Atom Word mode is on if this variable is nonzero. See section 22.1 [Minor Modes], page 115.

Auto Directory Display
: If this is nonzero, certain file operations automatically display the file directory. See section 13.4 [Directories], page 65.

Auto Fill Mode The minor mode Auto Fill mode is on if this variable is nonzero. See section 11.4 [Filling], page 50.

Auto Push Point Notification
: The value of this variable is the string printed in the echo area by some commands to notify you that the mark has been set to the old location of point. See section 10 [Search], page 41.

Auto Push Point Option (500)
: Searches set the mark if they move at least this many characters. See section 10 [Search], page 41.

Auto Save All Buffers
: If this is nonzero, auto save saves all buffers that are modified, not just the selected buffer. See section 13.3 [Auto Save], page 63.

Case Replace (1) When Case Replace is nonzero, Replace String and Query Replace attempt to preserve case when they replace. See section 19 [Replace], page 87.

Case Search (1) If Case Search is nonzero, searches of all sorts allow upper case letters and lower case letters to match each other. It controls the TECO flag FS BOTH CASE♦. See section 10 [Search], page 41.

Collapse in Comparison (nonexistent)
If this variable exists and is not zero, it should be a string of characters for M-X Compare Windows to regard as insignificant. See section 16 [Windows], page 77.

Comment Begin This is the string used to start new comments. If it is zero, the value of Comment Start is used. See section 20.5 [Comments], page 95.

Comment Column This is the column at which comments are aligned. See section 20.5 [Comments], page 95.

Comment End This is the string which is used to end comments. It is often empty for languages in which comments end at the end of the line. See section 20.5 [Comments], page 95.

Comment Multi Line (nonexistent)
If this variable exists and is nonzero, then when Auto Fill mode breaks a comment line, it does not insert a new comment starter on the new line. This is for use with languages that have explicit comment terminators, if you want single multi-line comments instead of single-line comments on consecutive lines. See section 20.5 [Comments], page 95.

Comment Rounding (/8+1*8)
This is the TECO program used to decide what column to start a comment in when the text of the line goes past the comment column. The argument to the program is the column at which text ends. See section 20.5 [Comments], page 95.

Comment Start This is the string used for recognizing existing comments, and for starting new ones if Comment Begin is zero. If Comment Start is zero, semicolon is used. See section 20.5 [Comments], page 95.

Compile Command (nonexistent)
If this variable exists and is nonzero, its value should be a TECO program to be used by the M-X Compile command to compile the file. See section 20.2 [Compile], page 92.

Compiler Filename (nonexistent)
If this variable exists and is nonzero, its value should be the name of the compiler to use, suitable for a colon command. By default, the name of the major mode is used as the name of the compiler. See section 20.2 [Compile], page 92.

Compiler Switches (nonexistent)
If this variable exists and is nonzero, its value is used as switches for compilation. See section 20.2 [Compile], page 92.

Cursor Centering Point (40)
This specifies how far from the top of the screen point ought to

instead of at the top of the screen. It controls the TECO flag FS ECHO ERRORS↑. See section 24.2 [Lossage], page 134.

Exit Hook (nonexistent)
If this variable exists and is nonzero, its value is a TECO program to be executed whenever EMACS is exited, instead of the normal action of doing an auto save. The subroutine & Exit EMACS is responsible for executing it. See section 6.4 [Exiting], page 27.

Exit to Superior Hook (nonexistent)
If this variable exists and is nonzero, its value is a TECO program to be executed whenever EMACS is about to return to its superior job, in addition to all normal actions.

Fill Column (70) The value of Fill Column is the width used for filling text. It controls the TECO flag FS ADLINE↑. See section 11.4 [Filling], page 50.

Fill Extra Space List (.?!)
The characters in this string are the ones which ought to be followed by two spaces when text is filled. See section 11.4 [Filling], page 50.

Fill Prefix The value of this variable is the prefix expected on every line of text before filling and placed at the front of every line after filling. It is usually empty, for filling nonindented text. See section 11.4 [Filling], page 50.

Indent Tabs Mode (-1)
If Indent Tabs Mode is nonzero, then tab characters are used by the indent commands. Otherwise, only spaces are used. See section 11.3 [Indenting Text], page 48.

<libname> Setup Hook (nonexistent)
If this variable exists and is nonzero, its value should be a TECO program to be executed when the library <libname> is loaded. The library's & Setup function is responsible for doing this. If the library has no & Setup function, it will not handle a setup hook either. See section 22.2 [Libraries], page 116.

<libname> Kill Hook (nonexistent)
Some libraries may execute the value of this variable, if it exists and is nonzero, when the library is being removed from core with Kill Libraries. This is done by the library's & Kill function; if the library has no & Kill <libname> Library function, it will not handle a kill hook. See section 22.2 [Libraries], page 116.

Lisp <function> Indent
This variable controls the indentation within calls to the function <function>. Actually, the variable used is not always Lisp <function> Indent, but rather <language> <function> Indent, where <language> is the value of Lisp Indent Language. See section 20.3 [Indenting], page 93.

Lisp Indent DEFanything (1)
The value of this variable controls indentation within calls to functions whose names start with "def". Actually, the variable used is not always Lisp Indent DEFanything, but rather <language>

PAGE Flush CRLF If this variable exists and is nonzero, the PAGE library expects every page to start with a blank line, which is not considered part of the contents of the page. See section 18.1 [PAGE], page 84.

Paragraph Delimiter (.↑O↑I↑O ↑O')
This is the TECO search string used to recognize beginnings of paragraphs. See section 11.2 [Sentences], page 46.

Permit Unmatched Paren (-1)
Controls whether the bell is run if you insert an unmatched close parenthesis. See section 20.4 [Matching], page 94.

Prefix Char List This variable's value is a string which lists all the prefix characters defined, so that self-documentation facilities can find any subcommands of prefix characters which call a given function. See the file INFO;CONV >, node Prefix.

Quote Execute Command (nonexistent)
If this variable exists and is zero, then M-X does not quote ↑] characters which appear in the string arguments of the command. See section 5.2 [Extended Commands], page 21.

Read Line Delay This is the amount of time, in 30'ths of a second, which EMACS should wait after starting to read a line of input, before it prompts and starts echoing the input.

Readable Word Abbrev Files (nonexistent)
If this variable exists and is nonzero, word abbrev files will be written in the format that M-X List Word Abbrevs uses, instead of in a less readable but faster loading format. See section 25.1.6 [Saving Abbrevs], page 146.

Region Query Size (5000)
Many commands which act on the region require confirmation if the region contains more than this many characters. See section 8 [Mark], page 31.

Return from Superior Hook (nonexistent)
If this variable exists and is nonzero, its value should be a TECO program to be executed whenever EMACS is resumed after being exited. See section 6.4 [Exiting], page 27.

SAIL Character Mode
If this is nonzero, characters in the buffer with ASCII codes 0 through 37 are displayed without conversion. Do not try to use this feature except on terminals specially equipped to handle it. The variable controls the TECO flag FS SAIL◊. See section I.1 [Ideal Keyboards], page 155.

Save Word Abbrevs (nonexistent)
If this variable exists, its value determines which abbrevs will be saved upon exit from EMACS when abbrevs have been modified. Setting it to 1 causes all abbrevs to be saved. See section 25.1.6 [Saving Abbrevs], page 146. Setting it to -1 causes just the incremental abbrevs to be saved. See section 25.2.4 [Dumped Environments], page 149.

Search Exit Char (27)

Underline End (nonexistent)
If this var able exists, its value should be the cha:acter or string to use to end underlines for the M-_ command. See section 11.7 [Underlining], page 53.

Visit File Hook (nonexistent)
If this variable exists and is nonzero, its value should be a TECO program to be executed whenever a file is visited. See section 13.1 [Visiting], page 61.

Visit File Save Old (1)
This variable controls whether visiting a file offers to save the file previously visited in the same buffer, if it has changes. See section 13.1 [Visiting], page 61.

WORDAB All Caps (nonexistent)
If this variable exists and is nonzero, expanding an all-upper-case abbrev to a multi-word expansion will cause the words in the expansion to be all-upper-case, instead of just having their first letters uppercased. See section 25.1.2 [Controlling Expansion], page 145.

WORDAB Ins Chars (nonexistent)
If this variable exists when Word Abbrev Mode is turned on, it should be a string containing precisely those characters which should terminate and expand an abbrev. This variable overrides Additional Abbrev Expanders (q.v.). See section 25.2.1 [Customizing WORDAB], page 147.

Non-Control Non-Meta Characters:

```
Backspace ^R Backward Character
Tab       ^R Indent According to Mode
Linefeed  ^R Indent New Line
Return    ^R CRLF
Altmode   ^R Prefix Meta
Rubout    ^R Backward Delete Character
```

Control Characters:

```
Alpha     ^R Complement SAIL Mode
Altmode   ^R Exit
Space     ^R Set/Pop Mark
%    ..   ^R Replace String
-    ..   ^R Negative Argument
0 thru 9  ^R Argument Digit
;    ..   ^R Indent for Comment
<    ..   ^R Mark Beginning
=    ..   What Cursor Position
>    ..   ^R Mark End
@    ..   ^R Set/Pop Mark
A    ..   ^R Beginning of Line
B    ..   ^R Backward Character
C    ..   ^R Prefix Control-Meta
D    ..   ^R Delete Character
E    ..   ^R End of Line
F    ..   ^R Forward Character
G    ..   ^R Quit
H    ..   ^R Backward Character
I    ..   ^R Indent According to Mode
J    ..   ^R Indent New Line
K    ..   ^R Kill Line
L    ..   ^R New Window
M    ..   ^R Self Insert for Formatting Character
N    ..   ^R Down Real Line
O    ..   ^R Open Line
P    ..   ^R Up Real Line
Q    ..   ^R Quoted Insert
R    ..   ^R Reverse Search
S    ..   ^R Incremental Search
T    ..   ^R Transpose Characters
U    ..   ^R Universal Argument
V    ..   ^R Next Screen
W    ..   ^R Kill Region
X    ..   is a prefix character.  See below.
Y    ..   ^R Un-kill
Z    ..   ^R Return to Superior
\    ..   ^R Prefix Meta
]    ..   Abort Recursive Edit
^    ..   ^R Prefix Control
Rubout    ^R Backward Delete Hacking Tabs
```

Meta Characters:

```
Linefeed  ^R Indent New Comment Line
Return    ^R Back to Indentation
Altmode   ^R Execute Minibuffer
#     ..  ^R Change Font Word
%     ..  ^R Query Replace
$     ..  ^R Correct Word Spelling
'     ..  ^R Upcase Digit
(     ..  ^R Make ()
)     ..  ^R Move Over )
-     ..  ^R Negative Argument
.     ..  ^R Find Tag
/     ..  ^R Describe
0 thru 9  ^R Argument Digit
;     ..  ^R Indent for Comment
<     ..  ^R Goto Beginning
=     ..  ^R Count Lines Region
>     ..  ^R Goto End
?     ..  ^R Describe
@     ..  ^R Mark Word
A     ..  ^R Backward Sentence
B     ..  ^R Backward Word
C     ..  ^R Uppercase Initial
D     ..  ^R Kill Word
E     ..  ^R Forward Sentence
F     ..  ^R Forward Word
G     ..  ^R Fill Region
H     ..  ^R Mark Paragraph
I     ..  ^R Tab to Tab Stop
J     ..  ^R Indent New Comment Line
K     ..  ^R Kill Sentence
L     ..  ^R Lowercase Word
M     ..  ^R Back to Indentation
N     ..  ^R Down Comment Line
P     ..  ^R Up Comment Line
Q     ..  ^R Fill Paragraph
R     ..  ^R Move to Screen Edge
S     ..  ^R Center Line
T     ..  ^R Transpose Words
U     ..  ^R Uppercase Word
V     ..  ^R Previous Screen
W     ..  ^R Copy Region
X     ..  ^R Extended Command
Y     ..  ^R Un-kill Pop
[     ..  ^R Backward Paragraph
\     ..  ^R Delete Horizontal Space
]     ..  ^R Forward Paragraph
^     ..  ^R Delete Indentation
_     ..  ^R Underline Word
~     ..  ^R Buffer Not Modified
Rubout    ^R Backward Kill Word
```

Control-Meta Characters:

```
Backspace ^R Mark Defun
Tab       ^R Indent for Lisp
Linefeed  ^R Indent New Comment Line
Return    ^R Back to Indentation
(     ..  ^R Backward Up List
)     ..  ^R Forward Up List
-     ..  ^R Negative Argument
0 thru 9  ^R Argument Digit
;     ..  ^R Kill Comment
?     ..  ^R Documentation
@     ..  ^R Mark Sexp
A     ..  ^R Beginning of Defun
B     ..  ^R Backward Sexp
C     ..  ^R Exit
D     ..  ^R Down List
E     ..  ^R End of Defun
F     ..  ^R Forward Sexp
G     ..  ^R Format Code
H     ..  ^R Mark Defun
I     ..  ^R Indent for Lisp
J     ..  ^R Indent New Comment Line
K     ..  ^R Kill Sexp
M     ..  ^R Back to Indentation
N     ..  ^R Forward List
O     ..  ^R Split Line
P     ..  ^R Backward List
Q     ..  ^R Indent Sexp
R     ..  ^R Reposition Window
T     ..  ^R Transpose Sexps
U     ..  ^R Backward Up List
V     ..  ^R Scroll Other Window
W     ..  ^R Append Next Kill
X     ..  ^R Instant Extended Command
[     ..  ^R Beginning of Defun
\     ..  ^R Indent Region
]     ..  ^R End of Defun
^     ..  ^R Delete Indentation
Rubout    ^R Backward Kill Sexp
```

Control-X is an escape prefix command with these subcommands:

```
^X ^B       List Buffers
^X ^C       ^R Return to Superior
^X ^D       ^R Directory Display
^X ^F       Find File
^X Tab      ^R Indent Rigidly
^X ^L       ^R Lowercase Region
^X ^N       ^R Set Goal Column
^X ^O       ^R Delete Blank Lines
^X ^P       ^R Mark Page
^X ^Q       ^R Set File Read-Only
^X ^S       ^R Save File
^X ^T       ^R Transpose Lines
^X ^U       ^R Uppercase Region
^X ^V       ^R Visit File
^X ^W       Write File
^X ^X       ^R Exchange Point and Mark
^X Altmode  ^R Re-execute Minibuffer
^X #        ^R Change Font Region
^X (        ^R Start Kbd Macro
^X .        ^R Set Fill Prefix
^X 1        ^R One Window
^X 2        ^R Two Windows
^X 3        ^R View Two Windows
^X 4        ^R Visit in Other Window
^X ;        ^R Set Comment Column
^X =        What Cursor Position
^X A        ^R Append to Buffer
^X B        Select Buffer
^X D        ^R Dired
^X F        ^R Set Fill Column
^X G        ^R Get Q-reg
^X H        ^R Mark Whole Buffer
^X I        ^R Info
^X K        Kill Buffer
^X L        ^R Count Lines Page
^X M        Send Mail
^X N        ^R Narrow Bounds to Region
^X O        ^R Other Window
^X P        ^R Narrow Bounds to Page
^X R        Read Mail
^X T        ^R Transpose Regions
^X W        ^R Widen Bounds
^X X        ^R Put Q-reg
^X [        ^R Previous Page
^X ]        ^R Next Page
^X ^        ^R Grow Window
^X _        ^R Underline Region
^X Rubout   ^R Backward Kill Sentence
```

Index

! 13, 87

% 59

& (in function names) 23
& Alter ..D 121
& Exit EMACS 28
& Mail Message 26

Permutation Table (buffer) 153
Permuted File (buffer) 153
Tags Search (buffer) 111
TAGS (buffer) 108, 110

--MORE-- 7

. 87

<cr> 10

@Begin 54
@End 54

Abbrev definition files 146, 149
Abbrev definition lists 143, 146, 148
Abbrevs 143
Abort Recursive Edit 118, 133
Aborting 133
Accumulator 103
Additional Abbrev Expanders 148
Address 103
Altmode 10, 11, 12, 20, 87, 88, 120, 130, 131
Append to File 39, 68
Apropos 29
Argument 144, 149
ASCII 9, 11
Atom Word mode 6, 46, 115
Auto Directory Display 65
Auto Fill 97
Auto Fill mode 6, 48, 50, 96, 115, 116, 127
Auto Push Point Notification 33
Auto Push Point Option 33
Auto Save All Buffers 65
Auto Save Default 63, 115
Auto Save Filenames 64
Auto Save Interval 65
Auto Save Max 65
Auto Save mode 6, 63, 115
Auto Save Visited File 64
Autoarg mode 18, 157
Autoloading 117

Backspace 11, 12
BARE library 23
Bit prefix characters 156
Blank lines 15, 36, 47, 96, 97
BOLIO 52
Bottom Display Margin 75
Buffers 7, 71, 78, 108, 111, 127
Buggestion 55
Bugs 140
Built-in functions 23

C- 9
C-. 110
C-; 95
C-< 32
C-> 32
C-@ 31
C-A 14, 47, 93
C-B 14, 89
C-C 10, 132
C-C C-Y 132
C-D 14, 20, 35
C-E 14, 47, 93
C-F 14
C-G 20, 27, 42, 61, 132, 133, 137
C-K 14, 35, 47
C-L 14, 75, 130, 161
C-M-(98
C-M-) 98
C-M-; 95
C-M-@ 32, 99
C-M-A 100, 103
C-M-B 54, 98
C-M-C 27, 66, 118
C-M-D 54, 98, 103
C-M-E 54, 100, 103
C-M-F 54, 98
C-M-G 54, 101, 104
C-M-H 32, 54, 100
C-M-K 35, 54, 98
C-M-L 83
C-M-M 48, 93
C-M-N 54, 98, 103

Continuation line 13
Control 9, 11, 138
Control characters, display of 12
Control characters, inserting 13
Control-Meta 98
Copy File 69
Correct Spelling 59
Count Matching Lines 88
Crashes 63
Create File 62
CRLF 12, 13
Cursor 5, 13
Cursor Centering Point 75
Customization 11, 23, 102, 123, 127, 147

Default Major Mode 71
Default Separator 160
Define Word Abbrevs 148
Defining abbrevs 144, 146
Defuns 32, 100
Delete File 69
Delete Matching Lines 88
Delete Non-Matching Lines 68, 88
Deletion 13, 35, 57, 88, 136
Describe 29, 118
Digit Shift Table 58
Directory 66, 68, 69
Directory Lister 65
DIRED 66, 117
Disasters 63
Display Matching Paren 94
Display Mode Line Inverse 7
Display Overprinting 12
Dissociated Press 55
Documentation 30
Dollar Sign 12
Down Picture Movement 151
Drastic Changes 63
Dumped environments 149

Echo area 5, 20, 51, 83, 144
Echo Area Height 5
Edit Indented Text 48
Edit key 156
Edit Options 26, 118
Edit Picture 27, 151
Edit Syntax Table 120
Edit Tab Stops 45, 49
Edit Word Abbrevs 146
Editor Type 6
End of Buffer Display Margin 75
Environments 53
Error handler 5
Error message 5
Error Messages in Echo Area 134
Escape key 155
EVARS 116, 123, 127
EVARS files 146, 147
Exit Hook 28

Exit to Superior Hook 28
Exiting 27
Exiting EMACS 146, 149
Expand Word Abbrevs in Region 145
Expander characters 148
Expanding abbrevs 145
Extended commands 19

FAIL 103
File dates 63
File deletion 66
File directory 65
File Versions Kept 66
Files 7, 14, 61, 63, 68, 79, 127
Fill Column 50, 115, 118
Fill Prefix 47, 51
Filling 50
Find File 72, 108
Find Pat 99
FLUSHED 7
Fonts 52
Formatting 48, 101
Formfeed 83
FS Flags 49, 75, 95, 104, 121, 142, 158
Functions 11, 19, 23

Global abbrevs 143, 147, 148
Grinding 101

Help 10, 29, 138
Home Directory 123, 137
Hooks 150

Ignoriginal 55
Incremental abbrev definition files 149
Indent Tabs Mode 49, 116
Indentation 48, 93, 95, 101
Init file 157
Init files 123, 146, 147
Insert Buffer 153
Insert File 68
Insert Page Directory 85
Insert Word Abbrevs 148
Insertion 13, 68, 83
Instant Command Prompt 21
INTER 97

Join pages 85
Journal files 137
Justification 50

Keyboard macros 6, 128
Kill All Word Abbrevs 147, 148
Kill Buffer 73
Kill Libraries 117, 135
Kill ring 37, 135
Kill Some Buffers 73, 135
Killing 35, 37, 46, 47, 57, 98, 136
Killing abbrevs 145, 146, 147, 148

MQREPL 111
Muddle mode 97

Name Kbd Macro 129
Narrowing 51, 81, 83, 87
Next File 110
Next Screen Context Lines 76
Numeric argument 144, 149
Numeric arguments 17, 21, 22, 36, 38, 46, 48, 50, 51, 52, 53, 63, 65, 75, 77, 78, 87, 96, 101, 115, 118, 132, 156, 157

Only Global Abbrevs 148
Options 118
Outragedy 55
Overprinting 12
Overwrite mode 6, 115

PAGE 84
Page Delimiter 47, 84
PAGE Flush CRLF 85
Pages 32, 47, 83, 84, 153
Paragraph Delimiter 47, 104
Paragraphs 32, 46, 50, 54, 97, 103, 153
Parentheses 45, 94, 120
Permit Unmatched Paren 94
Permute Pages From Table 153
Pictures 151
Point 5, 13
Prefix characters 10, 124, 127
Prepend to File 39, 68
Preventing abbrev expansion 145
Printing characters 13
Printing terminal 161
Prompting 5, 20
PURIFY library 104

Q-registers 39
Query Replace 59, 87, 130, 131
Quit 137
Quitting 42, 133
Quote Execute Command 22
Quoting 13, 148

R 52, 53
Read Command Prompt 20
Read Incremental Word Abbrev File 149
Read Mail 26
Read Word Abbrev File 146, 149
Read-Only Visiting 62
Readable Word Abbrev Files 146
Reap File 66
Recursive editing level 6, 26, 66, 73, 120, 133, 138, 146
Redefining abbrevs 144
Redefining commands 123, 127
Region 31, 37, 47, 52, 53, 69, 81, 83, 100, 101, 144, 153
Region Query Size 52

Rename Buffer 72
Rename File 69
Replace String 87
Replacement 87, 88
Replay Journal File 137
Restarting 136
Return 10, 11, 13, 61
Return from Superior Hook 28
Return, stray 12
Revert File 63, 64
Right Picture Movement 151
RMAIL 6, 26
Rubout 10, 11, 13, 14, 20, 35, 57, 87, 91, 97, 115, 130
Run Library 117

S-expressions 97, 120
SAIL Character Mode 116, 155
SAIL characters 155
Save All Files 72
Save Word Abbrevs 146, 149
Saving 61, 63
Saving abbrevs 146
Screen 5, 75
SCRIBE 53
Scrolling 75, 76, 77
Search Exit Char 42
Search Exit Option 42
Searching 41, 85, 87
Select Buffer 71
Send Mail 26
Sentences 46, 57
Set Key 23, 147
Set Variable 118
Set Visited Filename 62, 69
Short Display Size 159
Slow Search Lines 159
Slow Search Separator 160
SLOWLY Maximum Speed 160
Sort Lines 153
Sort Pages 153
Sort Paragraphs 153
Sorting 153
Space 10, 11, 20, 50, 59, 87, 130
Space Indent Flag 48
Spell Program 59
SPLIT 135
Split File 135
Split pages 85
SRCCOM 63, 68
Start Journal File 137
String arguments 21, 22, 65, 118
String Search 42
Submode 6
Subroutines 23
Subsystem 6
Syntax table 45, 46, 94, 97, 100, 119

Tab 11, 45, 48, 91, 93, 97, 101

↑R Copy Region 37
↑R Correct Word Spelling 59
↑R Count Lines Page 84
↑R CRLF 13
↑R Delete Blank Lines 15, 36
↑R Delete Character 35
↑R Delete Horizontal Space 36, 48, 93
↑R Delete Indentation 36, 48, 93, 101
↑R Directory Display 65
↑R DIRED 68
↑R Down Comment Line 96
↑R Down Environment 54
↑R Down List 98
↑R Down Real Line 14
↑R Edit Quietly 159
↑R End Kbd Macro 129
↑R End of Defun 100
↑R End of Environment 54
↑R End of Line 14
↑R Exchange Point and Mark 31
↑R Execute Kbd Macro 129
↑R Execute Minibuffer 131
↑R Exit 27, 118
↑R Extract Sublist 99
↑R Fill Paragraph 50
↑R Fill Region 50
↑R Format Code 101
↑R Forward Character 14
↑R Forward Environment 54
↑R Forward List 98
↑R Forward Paragraph 47, 103
↑R Forward Sentence 47
↑R Forward Sexp 98
↑R Forward TECO Conditional 104
↑R Forward Up List 98
↑R Forward Word 45
↑R Get Q-reg 39
↑R Go to AC Field 103
↑R Go to Address Field 103
↑R Go to Next Label 103
↑R Go to Previous Label 103
↑R Goto Beginning 14
↑R Goto End 14
↑R Goto Next Page 85
↑R Goto Page 84
↑R Goto Previous Page 85
↑R Grow Window 78, 132
↑R Incremental Search 41, 85
↑R Indent for Comment 95
↑R Indent for Lisp 97, 101
↑R Indent Nested 104
↑R Indent New Comment Line 96
↑R Indent New Line 48, 91, 93, 101
↑R Indent Region 48, 101
↑R Indent Rigidly 48
↑R Indent Sexp 101
↑R Insert () 99
↑R Insert Pagemark 85
↑R Instant Extended Command 21
↑R Inverse Add Global Word Abbrev 144, 147, 148
↑R Inverse Add Mode Word Abbrev 144, 147
↑R Join Next Page 85
↑R Kbd Macro Query 130
↑R Kill Comment 95
↑R Kill Global Word Abbrev 147
↑R Kill Line 35
↑R Kill Mode Word Abbrev 147
↑R Kill Region 35
↑R Kill Sentence 35, 47
↑R Kill Sexp 35, 98
↑R Kill Terminated Word 103
↑R Kill Word 35, 46
↑R Lowercase Region 52
↑R Lowercase Word 51, 58
↑R Mark Beginning 32
↑R Mark Defun 32, 100
↑R Mark End 32
↑R Mark Environment 54
↑R Mark Page 32, 83
↑R Mark Paragraph 32, 47
↑R Mark Sexp 32, 99
↑R Mark Whole Buffer 32
↑R Mark Word 32, 46
↑R Move Over) 99
↑R Move to Screen Edge 76
↑R Narrow Bounds to Page 81, 83
↑R Narrow Bounds to Region 81
↑R New Window 14, 75, 161
↑R Next Page 83
↑R Next Screen 76
↑R One Window 77
↑R Open Line 15, 17
↑R Other Window 77
↑R PAGE Widen Bounds 85
↑R Prefix Control 10
↑R Prefix Control-Meta 10
↑R Prefix Meta 10
↑R Previous Page 83
↑R Previous Screen 76
↑R Put Q-reg 39
↑R Query Replace 88, 131
↑R Quoted Insert 13
↑R Re-execute Minibuffer 21, 88, 132
↑R Reposition Window 76
↑R Return to Superior 27
↑R Reverse Search 41, 85
↑R Save File 14, 61, 72
↑R Scroll Other Window 77
↑R Set Comment Column 96
↑R Set File Read-Only 62
↑R Set Fill Column 50, 115
↑R Set Fill Prefix 51
↑R Set Screen Size 159
↑R Set/Pop Mark 31
↑R Slow Display I-Search 159
↑R Slow Reverse Display I-search 159
↑R Start Kbd Macro 129